12-8-2023

Study of augmentations on historical manuscripts using TrOCR

Erez Meoded
Mississippi State University, meoderezjb@gmail.com

Study of augmentations on historical manuscripts using TrOCR

By

Erez Meoded

Approved by:

Jingdao Chen (Major Professor)
Shahram Rahimi
Sudip Mittal
T. J. Jankun-Kelly (Graduate Coordinator)
Jason M. Keith (Dean, Bagley College of Engineering)

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

December 2023

Name: Erez Meoded

Date of Degree: December 8, 2023

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Jingdao Chen

Title of Study: Study of augmentations on historical manuscripts using TrOCR

Pages in Study: 63

Candidate for Degree of Master of Science

Historical manuscripts are an essential source of original content. For many reasons, it is hard to recognize these manuscripts as text. This thesis used a state-of-the-art Handwritten Text Recognizer, TrOCR, to recognize a 16th-century manuscript. TrOCR uses a vision transformer to encode the input images and a language transformer to decode them back to text. We showed that carefully preprocessed images and designed augmentations can improve the performance of TrOCR. We suggest an ensemble of augmented models to achieve an even better performance.

DEDICATION

I am dedicating this work to the world I knew as a child, to the open green fields full of life that were replaced with sun-burning concrete. To the books, people read while riding the public bus to their work. I am also dedicating this work to the next generations, who will have to work hard and collaboratively to restore this world to the same place I knew it when I was a child. My prayers, my guidance, and my hopes are with them. We trust you for your children!

ACKNOWLEDGEMENTS

"A Psalm of thanksgiving." (Psalm 100:1) to "The Lord is my shepherd" (Psalm 23:1[1]). I would like to thank the Mississippi State University for its excellent education system. The committee members Jingdao Chen (Major Professor), Shahram Rahimi, and Sudip Mittal. Special thanks to Professor Chen for his superb and close guidance. Last spring, I joined Professor Rahimi's seminar CSE 8011, where he gave the class all the knowledge on how to write a successful paper and encouraged us to write a thesis. Without his advice and enthusiasm, I would not write this thesis.

[1] Both verses are from https://tehilim-online.co.il/%D7%AA%D7%94%D7%99%D7%9C%D7%99%D7%9D

TABLE OF CONTENTS

APPENDIX

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Historical manuscripts are an essential source of original content that we can learn from and connect to other content. They are “regarded primarily if not exclusively as materials for research” [1]. Although researchers and librarians have made well-designed archives of historical manuscripts, only a few are in a digital, searchable format. Researchers have recently used Handwritten Text Recognition (HTR) algorithms to convert modern and historical manuscripts from scanned images into digital text files. The earliest HTR tools use simple imaging techniques, such as Optical Character Recognition (OCR) scripting [2], feature-based classification and clustering [3], and feature word locating [4]. Later models used Artificial Intelligence (AI), such as the Hidden Markov Model (HMM) [5], RNN [6], and CNN-RNN Hybrid Networks [7].

Advanced AI models improve the quality and performance of modern manuscripts HTR both in time of computation and accuracy. However, historical manuscripts introduce three significant challenges: the scarcity of transcriptions (labels), the language gap, and the diversity of writing styles. Modern AI models are preferably trained on an extensive collection of samples, usually millions. But while many scanned manuscripts exist, only a few have a reliable transcription. Large Language Models (LLM) cannot efficiently recognize historical manuscripts because of the language gap. While researchers train LLM on available modern language datasets, historical manuscripts convey a different language. Lastly, a sample of cursive writing taken from historical documents is very different from modern ones and other historical documents. These three challenges make training a current AI model to digitize historical manuscripts challenging.

The invention of transformer architecture [8] brought new directions to modeling language and vision. Transformer is a parallel encoder-decoder paradigm with attention and without the RNN building blocks. Its multi-head design and the availability of the encoder output to the decoder improve the performance of language models to scores never seen before. Later work (e.g., BERT [9]) trained the transformer on massive data and held the weights for later fine-tuned downstream tasks (transfer learning). Other works extend the transformer to vision (e.g., ViT [10]). This proposed work will use the transformer-based model, TrOCR [11], and augmentations to recognize historical Latin manuscripts from the 16th century. Augmentation and pre-trained transformer-based models can alleviate the challenges we presented previously. We will study the effectiveness of different augmentations on the performance of text recognition of historical manuscripts using TrOCR.

Our contributions are:

- We measure the effectiveness of different augmentations on the performance of TrOCR.
- We showed that combining different augmentations by an ensemble of voters improves the overall performance of TrOCR.
- We showed the effectiveness of the pluggable feature in TrOCR by recognizing Latin.

CHAPTER II

LITERATURE REVIEW

2.1 Attention

While classical HTR models are feature-based on cursive characters [2], modern HTR models learn to find connections between input parts, e.g., between words in the text and between parts of the image. This architecture is called “attention.”

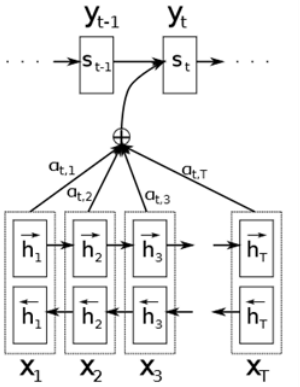


Figure 2.1 Alpha carries the combined connection between different parts of the sentence input *x* [10].

Attention improves the performance of classic Deep Neural Networks (DNNs) [12] by mimicking the human processing of language and vision, i.e., by finding connections between parts of the input [13]. The standard building blocks of these classic DNNs are RNN, LSTM, and GRU (Figure 2.1).

2.2 Transformer

Classic attention-based DNNs had sequential architecture (Figure 2.1). This property blocked the models from utilizing high-performance computing (GPUs) and limiting the attention mechanism's effectiveness. [8] introduce Transformer (Figure 2.4), which has an attention architecture without sequential building blocks (RNN). The transformer has two parts: an encoder and a decoder. The attention mechanism is implemented in different parts, in the multi-head self-attention and combining the encoder output and the decoder input. This innovative design and other hacks make the transformer better than previous models.

In the transformer paper [8], the authors noted two essential properties of this new architecture: parallelization and quadric complexity. The architecture of the transformer (Figure 2.4) enables processing input in parallel, which utilizes modern GPUs. On the other side, its Query, Key, and Value with SoftMax design (Figure 2.2) and the multi-head design (Figure 2.3) have a complexity of (2.1), making it infeasible for high-dimension input problems.

$$O(n^2 \cdot d) \tag{2.1}$$

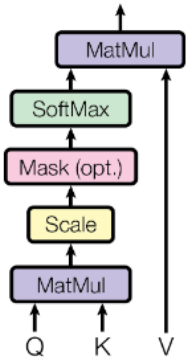


Figure 2.2 "Scaled Dot-Product Attention." The multiplication of the Q, K, and V matrices and the SoftMax standardization bring a quadratic complexity—image source [8].

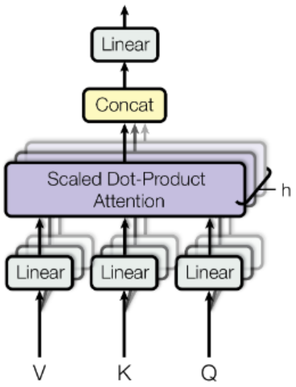


Figure 2.3 Multi-head (h) attention enables the transformer to process input in parallel and find better connections between different input parts—image source [8].

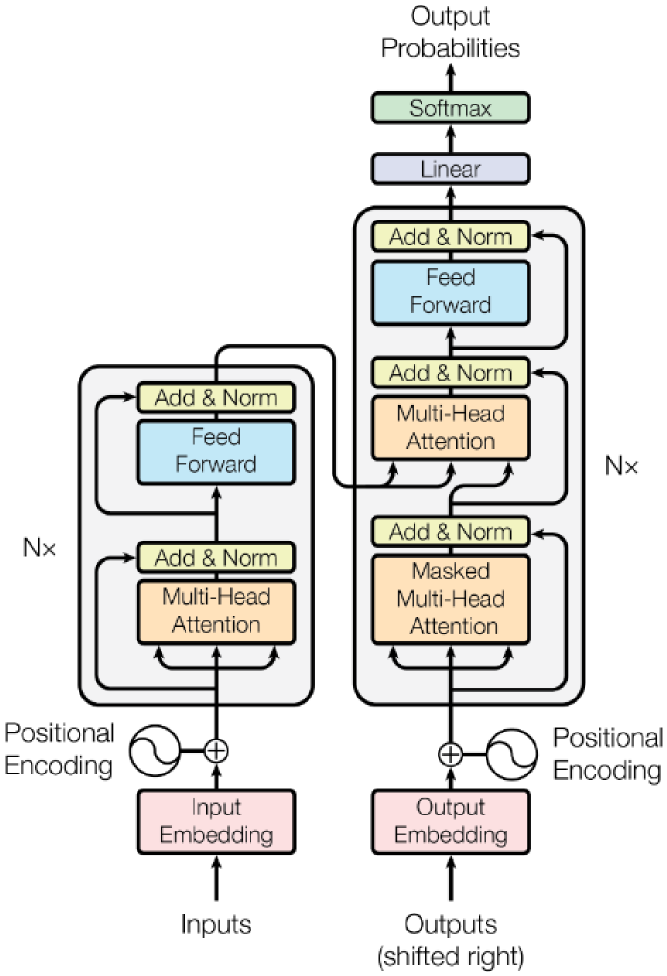


Figure 2.4 Transformer architecture. In comparison to Figure 2.1, there are no RNN blocks. The most crucial parts of this design are the attention head and the combined input to the decoder, the encoder output combined with the decoder output. The transformer is the base of most state-of-the-art modern models—image source [8].

2.3 Modern HTR

We can classify the current work on HTR as either attention-based HTR or transformer-based HTR. Recent work [14] suggests an encoder-attention-decoder HTR. Their model has multi-stage processing with ResNet to extract features and LSTM to predict from the previous stages. Previous similar work [15] did a grid search for the best hyperparameters on the encoder-decoder attention model from CNN and BLSTM layers.

2.4 Vision Transformer

The original transformer [8] can get input from embedded vocabulary with a relatively low dimension vector of size 512. Each vector represents a word in the language. Contrary to text input, in vision, the input dimension is exponentially more significant, where the model's input is images with multimillions of pixels on up to 3 (RGB) channels. Hence, the vision problems became intractable problems in the original transformer architecture. Vision Transformer (ViT) [10] (Figure 2.5) solves this dimensionality problem by inputting image patches instead of pixels without changing the original transformer!

One drawback of ViT is its lack of locality, which CNN has. The authors of ViT solve that by expanding the size of the training datasets to hundreds of millions of samples. Later works, for example, [16] suggest solving the lack of locality in ViT by prefixing ViT with CNN layers. This hybrid architecture achieved even better performance on a much smaller training dataset. The ViT encoder in the Huggingface's TrOCR implementation (which we will use in our study) also has one layer of Conv2d (input: image, output: 16x16 patches) before the ViT encoder.

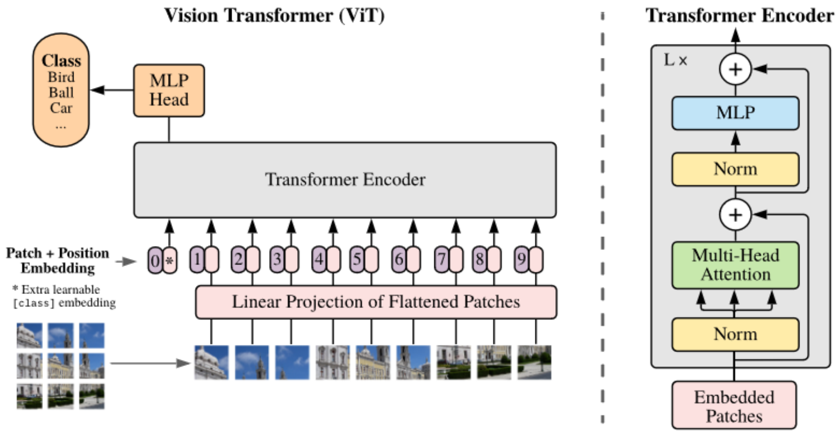


Figure 2.5 Vision Transformer (ViT) uses only the original transformer's encoder. It changes the input to patches of 16 by 16 pixels. ViT reduces the input dimension by splitting the image into patches of size 16x16 pixels (Image source [10])

2.5 TrOCR

TrOCR[2] [11] is a state-of-the-art (SOTA) complete transformer (with encoder and decoder) to recognize English text from images. It is a multi-transformer model with two pre-trained variants of ViT [10] and BERT [9]; the former is an image encoder, and the latter is a language decoder. We will use the Huggingface [17] implementation of TrOCR. In Huggingface, DeiT [17] is the image encoder in this implementation, and XLM-RoBERTa [18] is the text decoder. XLM-RoBERTa is a multilingual model. Its vocabulary has 100 languages.

TrOCR was trained in two steps: pre-training and fine-tuning. The pre-training step had two stages. In Stage 1, TrOCR was trained on 684M text lines from online PDF files. In Stage 2, it was trained on 17.9M synthetic handwritten text lines. In the second step, the authors fine-tuned the stage 2 model to four downstream OCR tasks: printed, handwritten, receipt, and scene.

[2] The source code of TrOCR is available on https://github.com/microsoft/unilm/tree/master/trocr

To fine-tune and benchmark, they used standard datasets. Since these datasets are small, the authors used six different augmentations in addition to keeping the original image. Figure 4.1 presents the augmentation for the handwritten task. TrOCR is available to the public in three sizes: small, base, and large, with different layers, hidden dimensions, and heads. Microsoft published twelve fine-tuned models and three pre-trained stage 1 models. It did not publish the pre-trained stage 2 models[3]. In this work, we will study the effectiveness of augmentations on TrOCR$_{\text{BASE}}$.

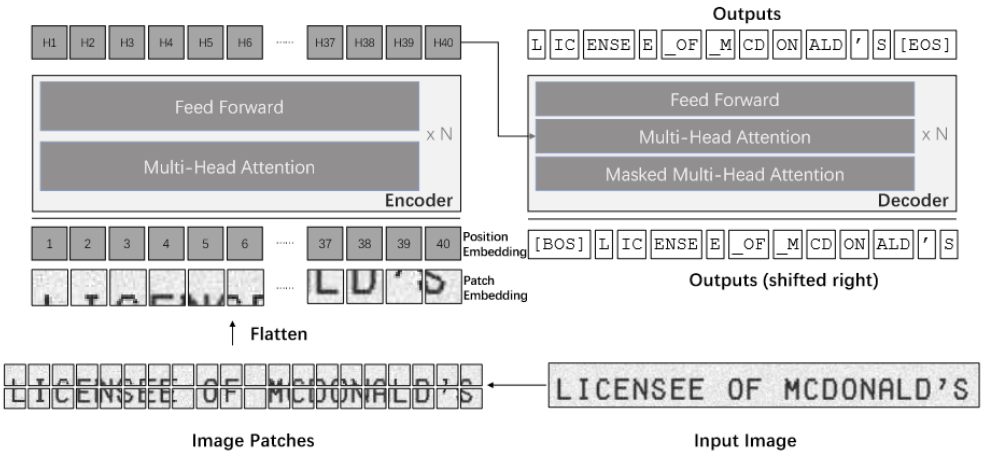


Figure 2.6 TrOCR Architecture. The data flow is clockwise from the bottom right to the top right. Base TrOCR has N of 12 and a vector of size 1024. The top left is the ViT encoder, and the top right is the original RoBERTa [19] decoder. (Image source [11])

TrOCR has a ViT encoder that accepts line images as input (Figure 2.6). ViT splits the image into patches of 16x16 pixels. Then, TrOCR processes the patches, similar to the original transformer processes text.

[3] https://github.com/microsoft/unilm/issues/831#issuecomment-1223778956

The original TrOCR could recognize only an English dataset and could not identify other languages efficiently[4]. To expand TrOCR to recognize different languages, Huggingface's TrOCR model replaces the original single language (English) RoBERTa decoder with an XLM-RoBERTa decoder pre-trained on one hundred languages, including Latin. This modification, as shown in this work, makes the TrOCR capable of recognizing Latin text images efficiently.

2.6 Augmentation in HTR

Augmentation is a helpful technique for creating synthetic samples when there are insufficient samples [11] or improving the overall model performance [20]. It distorts the input images to new ones the model has never seen [21]. Augmentation is widely used in HTR to alleviate the scarcity of transcripted manuscripts and improve HTR models' performance [22]. Common image distortions in HTR are rotation, translation, scaling, shearing (affine transform), grayscale erosion and dilation, rotations, skewing, and Gaussian noise [23], [24], [25], [26], blur, random noise, random stretch [22]. This work will study the six augmentations from the source code of TrOCR and the four augmentations we developed (see CHAPTER IV for technical information).

2.7 Character Error Rate (CER)

CER is a standard metric to evaluate the performance of text recognition algorithms. We will follow [27] and [28] and use CER as a performance metric. CER equation is a comparison between *prediction* and *reference* character by character, in the following formula (2.2):

$$CER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C} \tag{2.2}$$

[4] https://github.com/microsoft/unilm/issues/500

Table 2.1 gives examples of how to measure the number of substitutions (S), deletions (D), and insertions (I). These three are operations to transform the predicted text to the label text. After summating these three numbers, we divide them by the total number of characters in the label. Hence, CER is the rate (or percentage) between two values: the number of operations and characters.

Table 2.1 The Character Error Rate (CER) calculation consists of three basic parts: Substitution, Deletion, and Insertion.

	Reference	**Predictions**	
Substitution - S	ABCD	ABC**E**, A**YU**D	S = 1 in the first example and 2 in the second
Deletion - D	ABCD	ABC, ACD	D = 1 in both examples
Insertion - I	ABCD	ABCD**E**, AB**O**CD	I = 1 in both examples

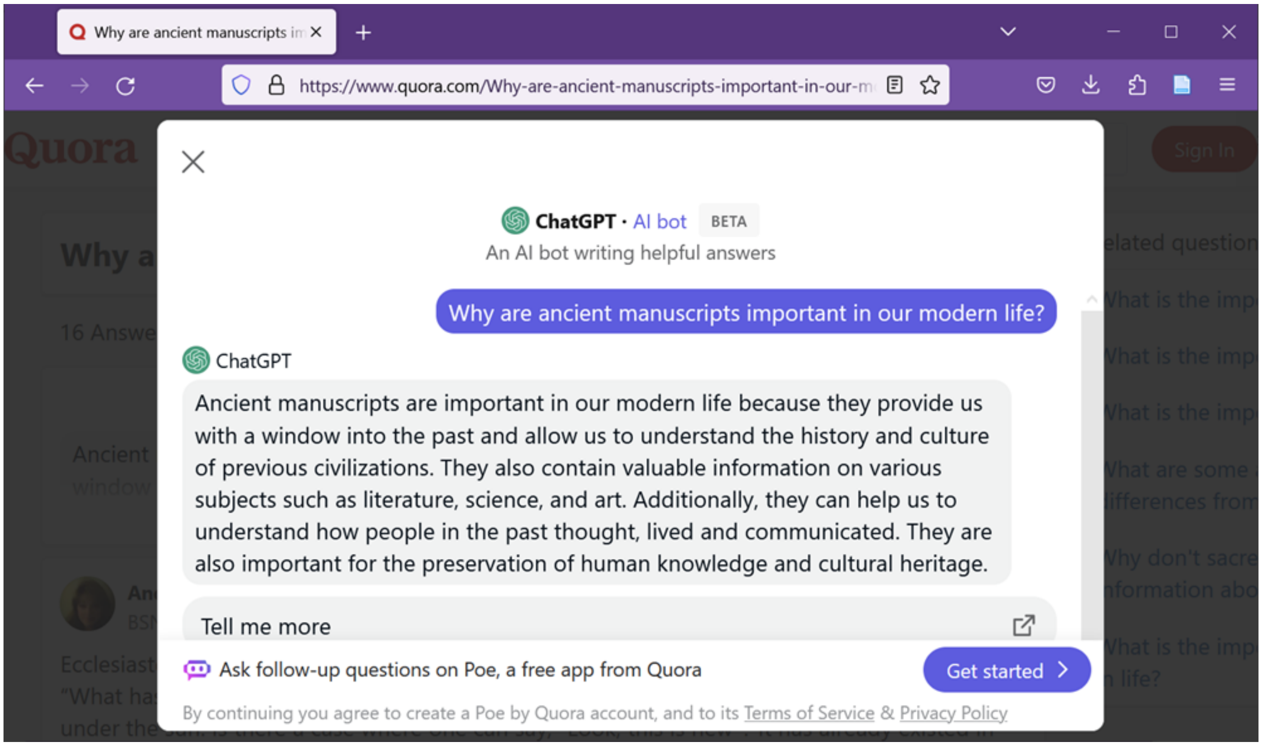
C is the number of correct characters
N is the number of characters in the *true* label (*reference*)
Another way to calculate N is simply by $S + D + C$.

2.8 Historical manuscripts

[1] defines historical manuscripts as:

"Records of historical value, written by hand or typewriter or its equivalent (as distinguished from printed records), in single or multiple forms."

Historical manuscripts present many challenges to HTR: Lack of transcription, distorted alignment [29], variation of cursive writing, quality of writing, language and culture gaps, and other challenges. Nevertheless, they are an essential source of original information for research [1] that holds particular values [30]. ChatGPT returns an engaging text about the importance of historical manuscripts (see Figure 2.7).


Why are ancient manuscripts im
https://www.quora.com/Why-are-ancient-manuscripts-important-in-our-m
Quora
Sign In
ChatGPT · AI bot BETA
An AI bot writing helpful answers

Why are ancient manuscripts important in our modern life?

ChatGPT

Ancient manuscripts are important in our modern life because they provide us with a window into the past and allow us to understand the history and culture of previous civilizations. They also contain valuable information on various subjects such as literature, science, and art. Additionally, they can help us to understand how people in the past thought, lived and communicated. They are also important for the preservation of human knowledge and cultural heritage.

Tell me more
Ask follow-up questions on Poe, a free app from Quora
Get started
By continuing you agree to create a Poe by Quora account, and to its Terms of Service & Privacy Policy

Figure 2.7 ChatGPT answers: “Why are ancient manuscripts important in our modern life?” Retrieved from quora.com on July 27, 2023

CHAPTER III

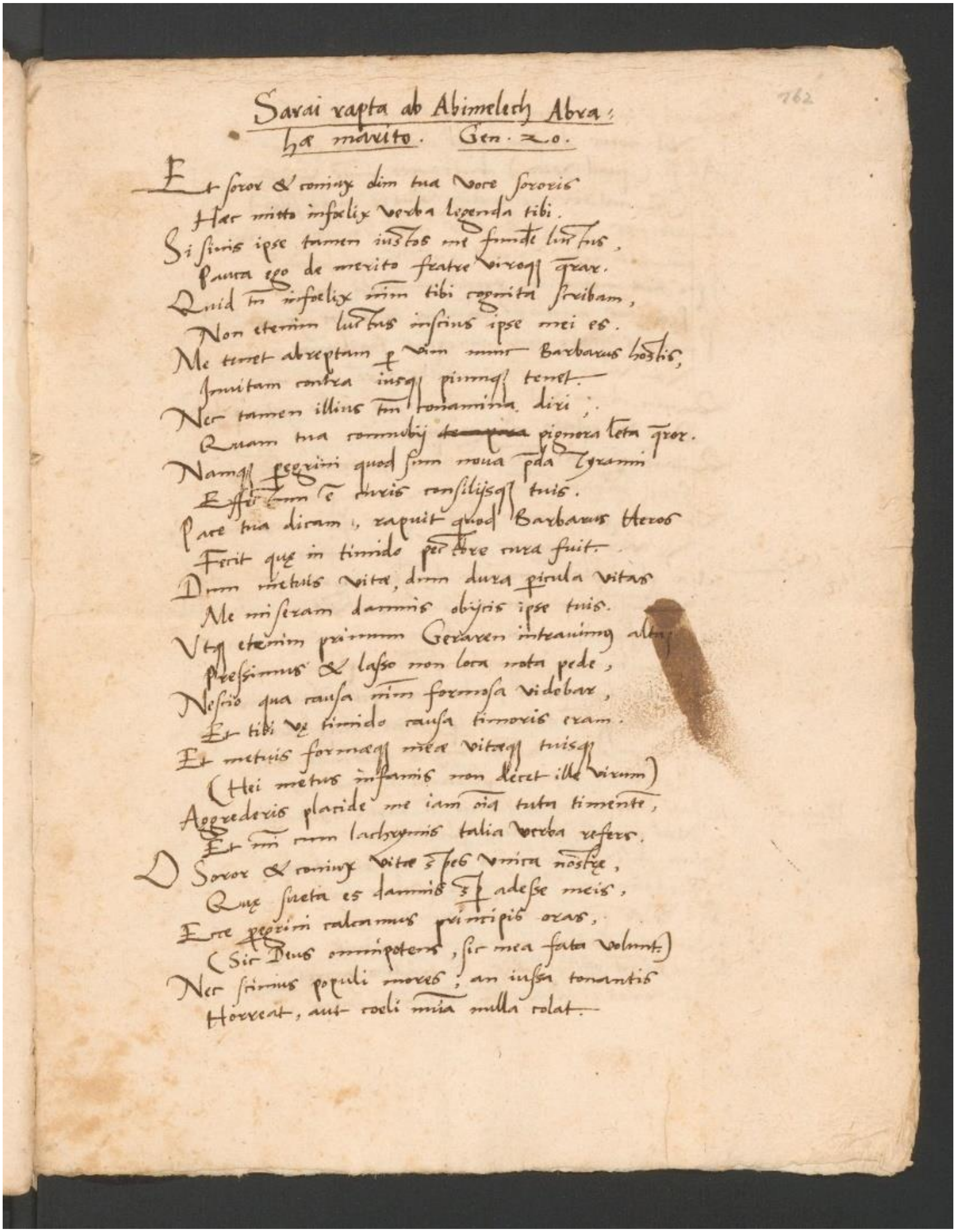
DATASET

3.1 Introduction

Following [28], we will study Rudolf Gwalther's manuscripts from the 16th century. Gwalther was a pastor and the head of the Reformed Church of Zurich in the 16th century. The manuscript is available at https://www.e-manuscripta.ch/zuz/doi/10.7891/e-manuscripta-26750 [31]. The authors of [28] used an AI-powered public service, Transkribus [32] (https://readcoop.eu/transkribus/), to recognize the images and made the output available to download at https://zenodo.org/record/4780947. We will note this output later as "the dataset."

3.2 Challenges

Figure 3.1 contains one original page from the dataset. We can see that this example page has stains, deletions with scribbling, upward curvature writing, and up and down line crossing. Also, the writing style differs from the modern style: frequent character joining, mixing calligraphy with simple scripts, background color, quality of paper, and more. These challenges require the HTR developer's particular attention to recognize the text. Also, it requires careful preparation of the input images and transforming them into the format the algorithm expects.

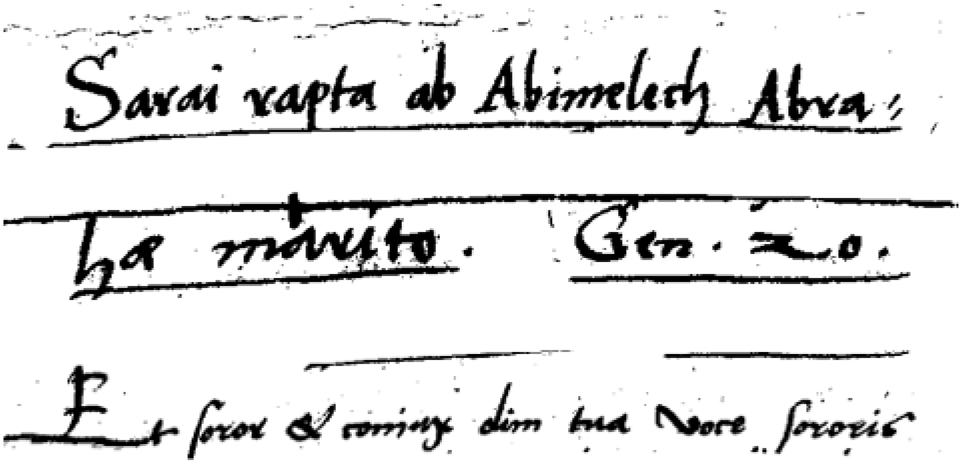


762

Sarai rapta ab Abimelech Abra:
ha marito. Gen. 20.

Et ſoror & coniux dum tua voce ſororis
Hac mitto infoelix verba legenda tibi.
Si ſciris ipſe tamen iuſtos me fronde luctus,
Pauca ego de merito fratre viroque querar.
Quid tu infoelix nimium tibi cognita ſcribam,
Non etenim luctus inſcius ipſe mei es.
Me terret abreptam per vim nunc Barbarus hoſtis,
Inuitam contra iusque piumque tenet.
Nec tamen illius tantum connubia dixi;
Quam tua connubij ~~[illegible]~~ pignora certa querar.
Namque peregrini quod ſum noua praeda Tyranni
Effectum e dictis conſilijsque tuis.
Pace tua dicam, rapuit quod Barbarus Heros
Fecit quae in timido pectore cura fuit.
Dum metuis vitae, dum dura pericula vitas
Me miſeram damnis obijcis ipſe tuis.
Vtque etenim primum Geraren intrauimus altam
Preſſimus & laſſo non loca nota pede,
Neſcio qua cauſa nimium formoſa videbar,
Et tibi ut timido cauſa timoris eram.
Et metuis formaeque meae vitaeque tuisque
(Hei metus infamis non decet ille virum)
Aggrederis placide me iam omnia tuta timentem,
Et mihi cum lachrymis talia verba refers.
O Soror & coniunx vitae spes vnica noſtrae,
Quae ſueta es damnis ſemper adeſſe meis,
Ecce peregrini calcamus principis oras,
(Sic Deus omnipotens, ſic mea fata volunt.)
Nec ſcimus populi mores; an iuſſa tonantis
Horreat, aut coeli numina nulla colat.

Figure 3.1 Original page from the source dataset (file 1111637 in the dataset).



Sarai rapta ab Abimelech Abra

ha marito. Gen. 20.

Et ſoror & coniux dum tua vere ſororis

Figure 3.2 The first three lines of Figure 3.1, after preprocessing

3.3 Preprocessing and dataset split

The dataset has 142 full-page images. Each page has an XML metadata file (APPENDIX D) with the coordinates of each line in the image and the label. The XML file is the output from Transkribus and includes the coordinates and the AI-generated text of 4,037 lines. We used the coordinates to crop the images into line images (see the code in APPENDIX C). TrOCR accepts *only* line images, i.e., images with a single line of text. Figure 3.1 shows an entire page file, and Figure 3.2 shows its corresponding first three line images after preprocessing. We followed [28] and split the data set into 3,603 lines for training and 433 lines for validation. We split the dataset into training and validation datasets using Scikit-Learn's [33] *train_test_split* API.

CHAPTER IV

AUGMENTATIONS

4.1 Introduction

Usually, it is expensive to acquire labeled images. Hence, researchers are at risk of getting biased models. To alleviate this problem, researchers apply augmentation (distortion) to the original images to create synthetic images [34]. Augmentation is an essential tool in the toolset of a researcher. It helps lower the bias by training the model on more extensive datasets with images the model did not see. In this work, we will study the effect of different augmentations on the performance of TrOCR in text-recognizing historical manuscripts.

4.2 The augmentations we used in our experiment

Our study includes ten different augmentations in addition to the baseline model. The developers of TrOCR used six augmentations to create millions of synthetic handwritten samples in the training phase. We adopted these six augmentations without changing the code except for fixing minor bugs in the TrOCR's augmentation code. So, our codes TrOCR's augmentation code is identical. In addition to these six augmentations, we developed four more augmentations that we think can improve handwritten text recognition performance. See Figure 4.1 and APPENDIX B.

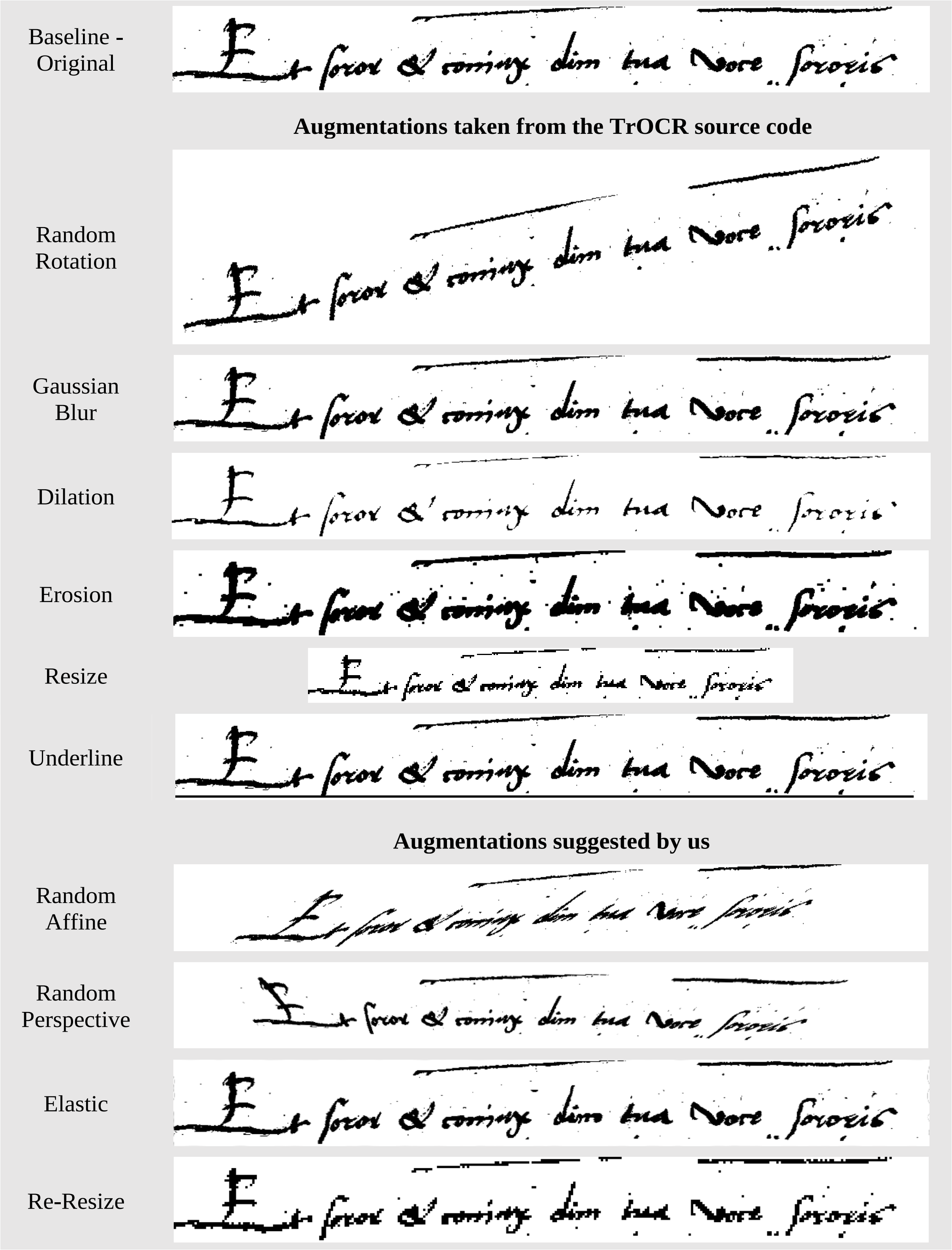


Figure 4.1 Ten augmentation models and baseline model. Elastic example in Figure 5.2

CHAPTER V

EXPERIMENT

5.1 Training

In our experiment, we trained eleven models, one model for each of the ten augmentations and one baseline model. In each of the first ten models, the data loader chooses uniformly (p=0.5) either to load the image as is or to augment it before loading it. This practice saves storage and is common in practice. We did not mix augmentations in one model. We trained the baseline model on the original data without any augmentation. The purpose of the baseline model is to be a benchmark for the augmentation models. The training ran for twenty epochs; however, we logged all the training and validation losses and the CER metric for each epoch. We validate the model on unseen samples. We followed [28] and used the Character Error Rate (CER) as the benchmark score. The training and validation cross-entropy [11] losses are in APPENDIX E.

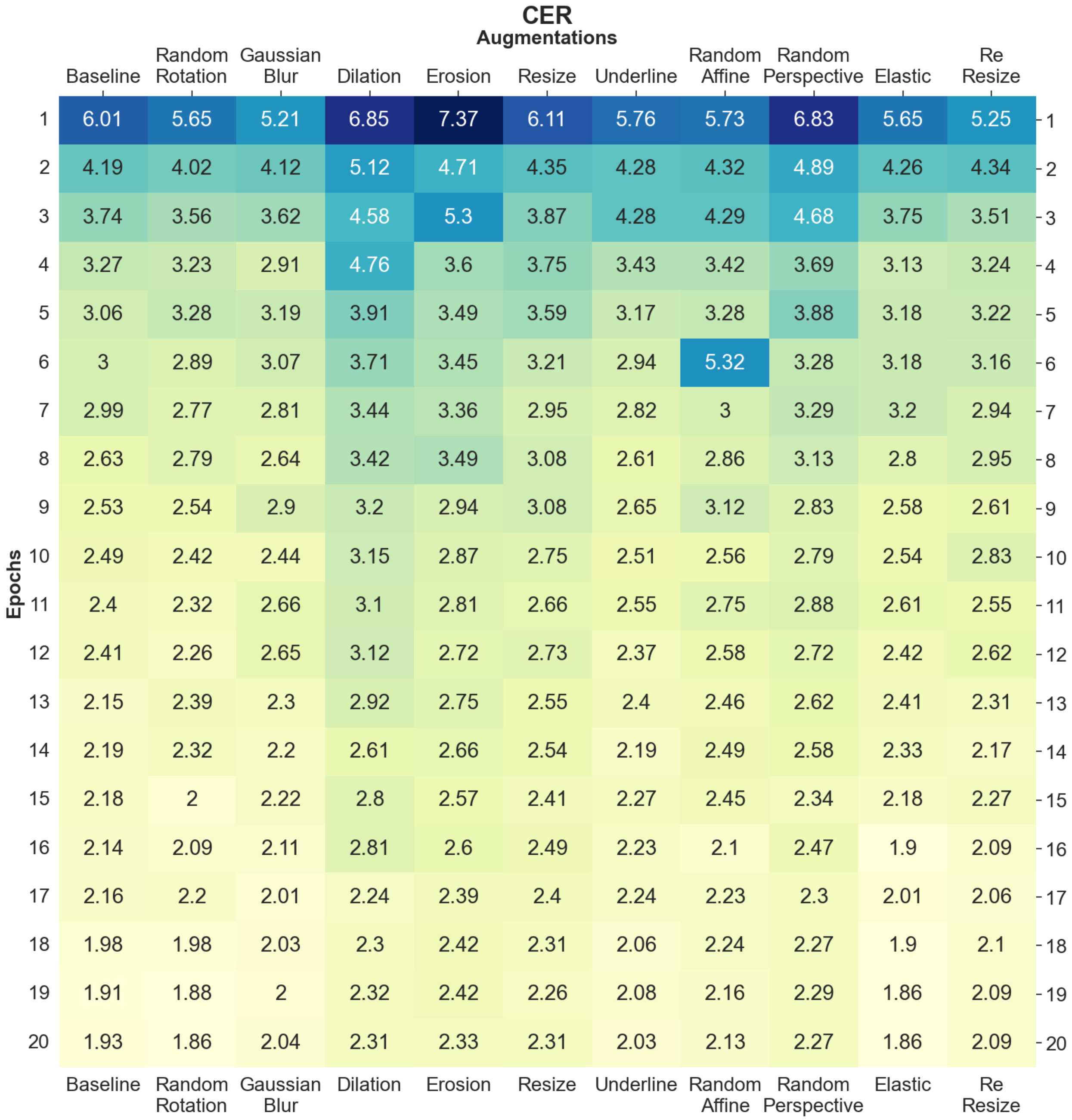
We wrote the training code in Python and used the PyTorch API and Huggingface platform to load the TrOCR pre-trained base model and train it. Training the $\text{TrOCR}_{\text{BASE}}$ requires a modern GPU with at least 16 Gigabytes (GB) of memory. We loaded the dataset and the code on Google's Colab cloud with the Pro+ plan. The training was on a single A100 GPU. It took about one hour to train and validate twenty epochs and about 15 minutes for five epochs. Since Colab is a share-based computation platform that allocates computation resources dynamically, we did not include the exact training time for each model as it varied considerably from run to run.

We kept the hyperparameters similar in all models' fine-tuning. In practice, if the inputs of the fine-tuning and pre-training close (but not equal), it is better to fine-tune using similar (but not identical) hyperparameters used in the original pre-training phase. Following this, we used the same hyperparameters the developers of TrOCR used to train the handwritten model. The training source code and a complete list of the hyperparameters are in APPENDIX A.

5.2 Results

The following results are from the validation dataset. We did not tweak the hyperparameters, nor did we change them between models, and following [28], we used the validation dataset as our test dataset and compared the models based on it. The results are from the TrOCR$_{\text{BASE}}$ model.

Figure 5.1 holds the Character Error Rate (CER) for each augmentation and each epoch. The scores are generally improving with the training. At the end of the fine-tuning, only two augmentation models are better than the baseline model, while all other models have higher CER.



Epochs	Baseline	Random Rotation	Gaussian Blur	Dilation	Erosion	Resize	Underline	Random Affine	Random Perspective	Elastic	Re Resize
1	6.01	5.65	5.21	6.85	7.37	6.11	5.76	5.73	6.83	5.65	5.25
2	4.19	4.02	4.12	5.12	4.71	4.35	4.28	4.32	4.89	4.26	4.34
3	3.74	3.56	3.62	4.58	5.3	3.87	4.28	4.29	4.68	3.75	3.51
4	3.27	3.23	2.91	4.76	3.6	3.75	3.43	3.42	3.69	3.13	3.24
5	3.06	3.28	3.19	3.91	3.49	3.59	3.17	3.28	3.88	3.18	3.22
6	3	2.89	3.07	3.71	3.45	3.21	2.94	5.32	3.28	3.18	3.16
7	2.99	2.77	2.81	3.44	3.36	2.95	2.82	3	3.29	3.2	2.94
8	2.63	2.79	2.64	3.42	3.49	3.08	2.61	2.86	3.13	2.8	2.95
9	2.53	2.54	2.9	3.2	2.94	3.08	2.65	3.12	2.83	2.58	2.61
10	2.49	2.42	2.44	3.15	2.87	2.75	2.51	2.56	2.79	2.54	2.83
11	2.4	2.32	2.66	3.1	2.81	2.66	2.55	2.75	2.88	2.61	2.55
12	2.41	2.26	2.65	3.12	2.72	2.73	2.37	2.58	2.72	2.42	2.62
13	2.15	2.39	2.3	2.92	2.75	2.55	2.4	2.46	2.62	2.41	2.31
14	2.19	2.32	2.2	2.61	2.66	2.54	2.19	2.49	2.58	2.33	2.17
15	2.18	2	2.22	2.8	2.57	2.41	2.27	2.45	2.34	2.18	2.27
16	2.14	2.09	2.11	2.81	2.6	2.49	2.23	2.1	2.47	1.9	2.09
17	2.16	2.2	2.01	2.24	2.39	2.4	2.24	2.23	2.3	2.01	2.06
18	1.98	1.98	2.03	2.3	2.42	2.31	2.06	2.24	2.27	1.9	2.1
19	1.91	1.88	2	2.32	2.42	2.26	2.08	2.16	2.29	1.86	2.09
20	1.93	1.86	2.04	2.31	2.33	2.31	2.03	2.13	2.27	1.86	2.09

Figure 5.1 CER score per augmentation. The first column is the baseline model. It is our benchmark model. The six augmentations next to the baseline are models with augmentations from the original handwritten TrOCR. The last four columns (on the right) are models from augmentations developed by us.

5.3 Analysis

As said before, compared to the baseline model, we have two classes of performance: augmentations with better CER scores and augmentations with worse CER scores. We can sort them as in Table 5.1. Looking back on Figure 4.1, we can see that augmentations with visibly lower distortion have a faster conversion to a minimum and better CER scores than other augmentations with higher visible distortion.

Table 5.1 Comparison between different augmentations on the last epoch (20). The source of the augmentation is either from the TrOCR training code, from our work, or as a benchmark.

epoch	**Source**	**Cer**	**Augmentation**
20	TrOCR	1.86	Random Rotation
20	Our	1.86	Elastic
20	Benchmark	1.93	Baseline
20	TrOCR	2.03	Underline
20	TrOCR	2.04	Gaussian Blur
20	Our	2.09	Re Resize
20	Our	2.13	Random Affine
20	Our	2.27	Random Perspective
20	TrOCR	2.31	Dilation
20	TrOCR	2.31	Resize

The validation loss in Figure 5.3, in the higher range of epochs (above 10), is in a plateau (convergence), which gives a clue about the pre-loss stage. To better interpret the findings, we omitted the training loss from the plots; we supplied them in APPENDIX E.

We can see an interesting (but not uncommon) phenomenon in Figure 5.3. At the same time, the cross-entropy loss of TrOCR is stable or even increases slightly in the higher epochs, and the CER metric continues to improve slightly. One possible explanation is that cross-entropy and CER are not fully correlated and measure the dissimilarity between True labels and predictions differently. Cross entropy measures the similarity of two probability distributions. In our case, between the probability distributions of the True labels and the predict labels. In contrast, CER calculates the rate of operations (insert, delete, and replace) needed to transform one string to another. Another possible explanation is the stage of the training, and the CER is expected to worsen in further epochs behind the twentieth.

Another way to plot the comparison between different augmentations is in Figure 5.4, where we can see significant improvement in the first 15 epochs and close to convergence in the last five epochs in all augmentations. The convergence can be seen clearly in Figure 5.5 and Figure 5.5. These two last plots prove the superiority of some augmentations over all other ones: Elastic (our development) and Random Rotation (TrOCR source code). It is possible to explain their better performance if we look at Figure 3.1; the handwriting of Gwalther has rows with constant upward curvature (like convex), i.e., on the left part of the line, the direction of the line is upward, the middle is flattened, and the right part is down direction. Random Rotation imitates this phenomenon in Gwalther's handwritten and supports the training. Similarly, Elastic augmentation supports the training by gently imitating the natural imperfection of the paper, pen, ink, and writer's hand. See Figure 5.2.

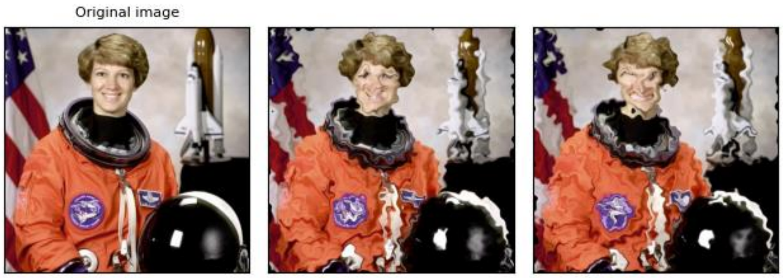


Figure 5.2 Example of Elastic Augmentation. Image source pytorch.org[5]

[5] https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#elastictransform

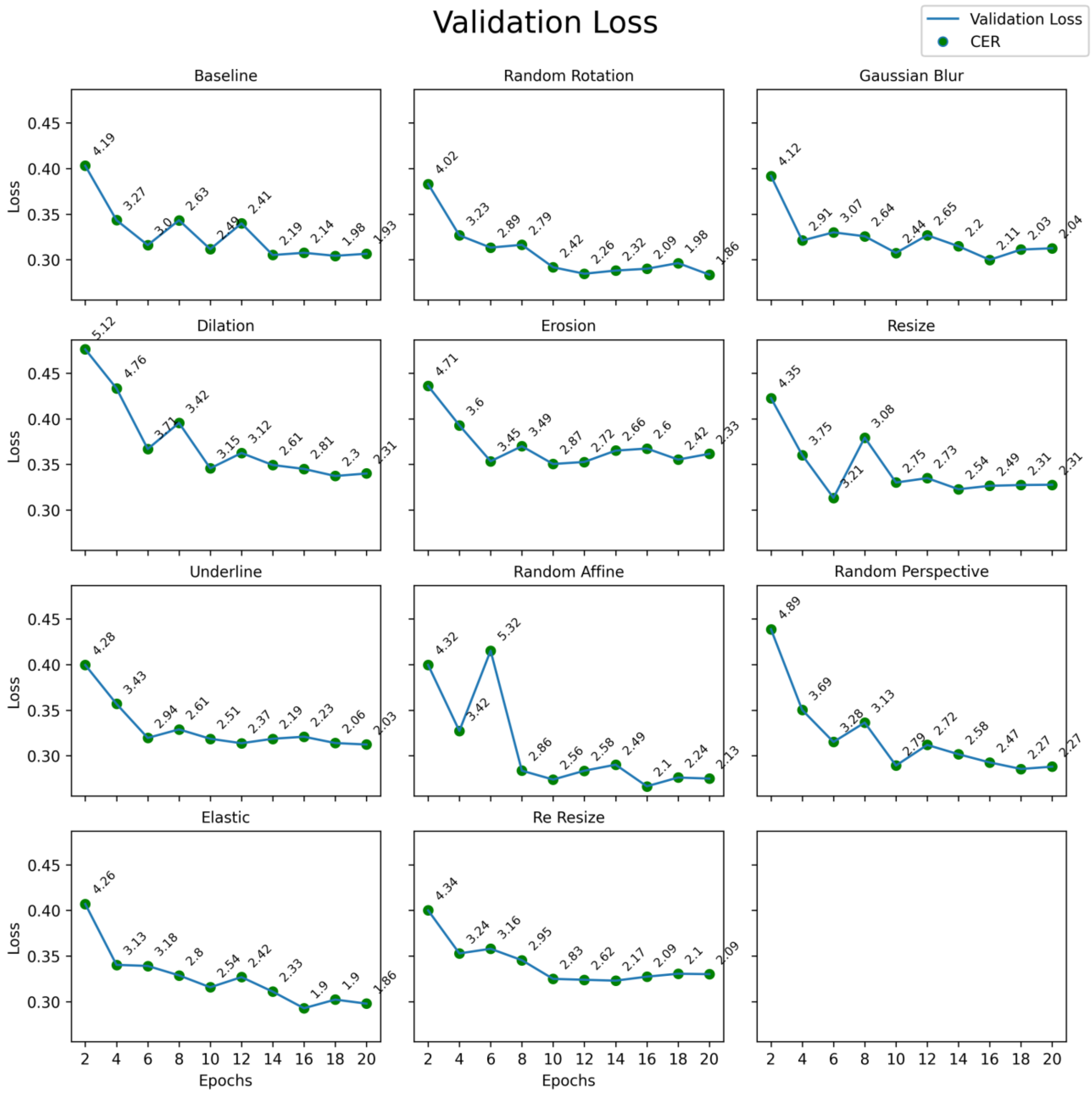


Figure 5.3 Validation loss for each even epoch. The labels are the CER score for each epoch.

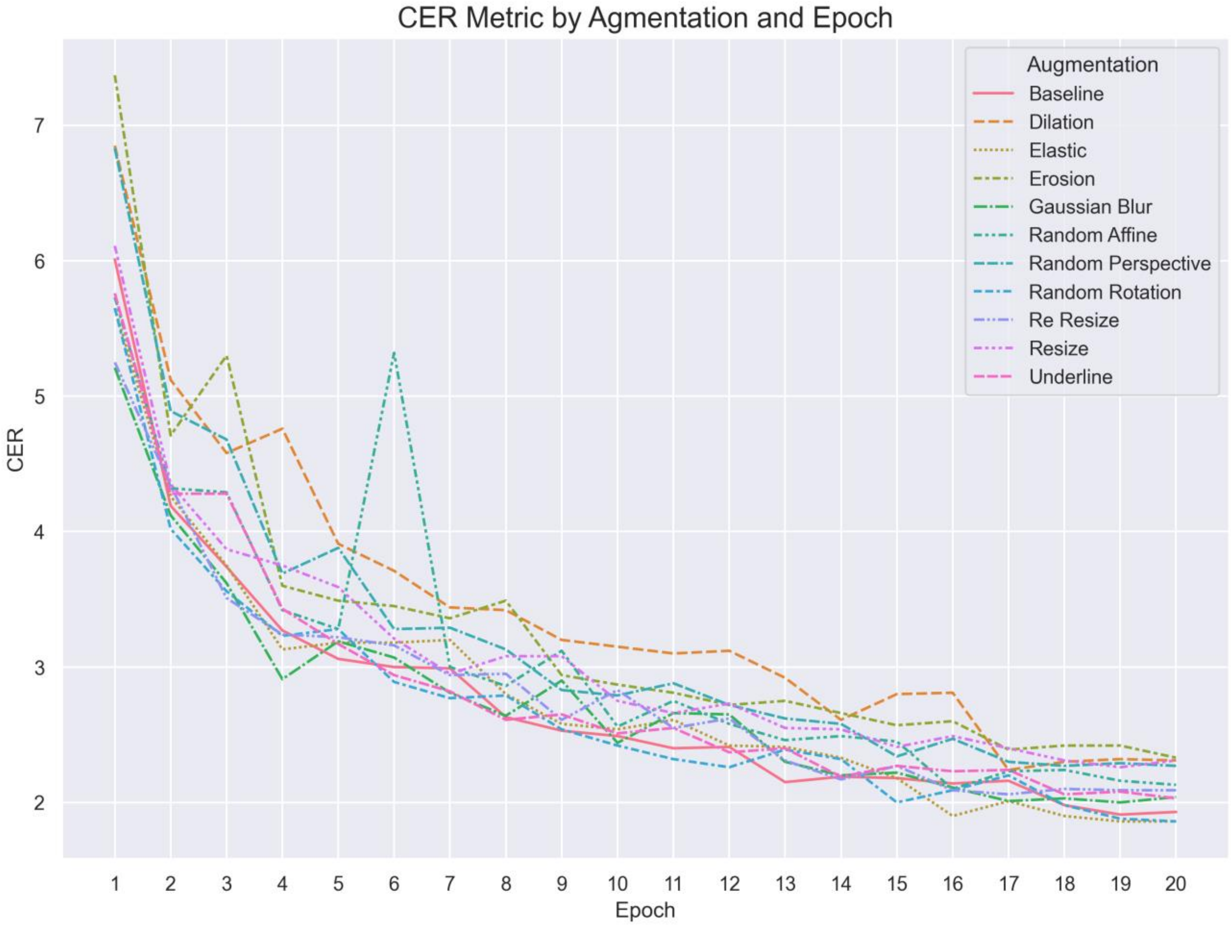


Figure 5.4 CER for each augmentation and each epoch.

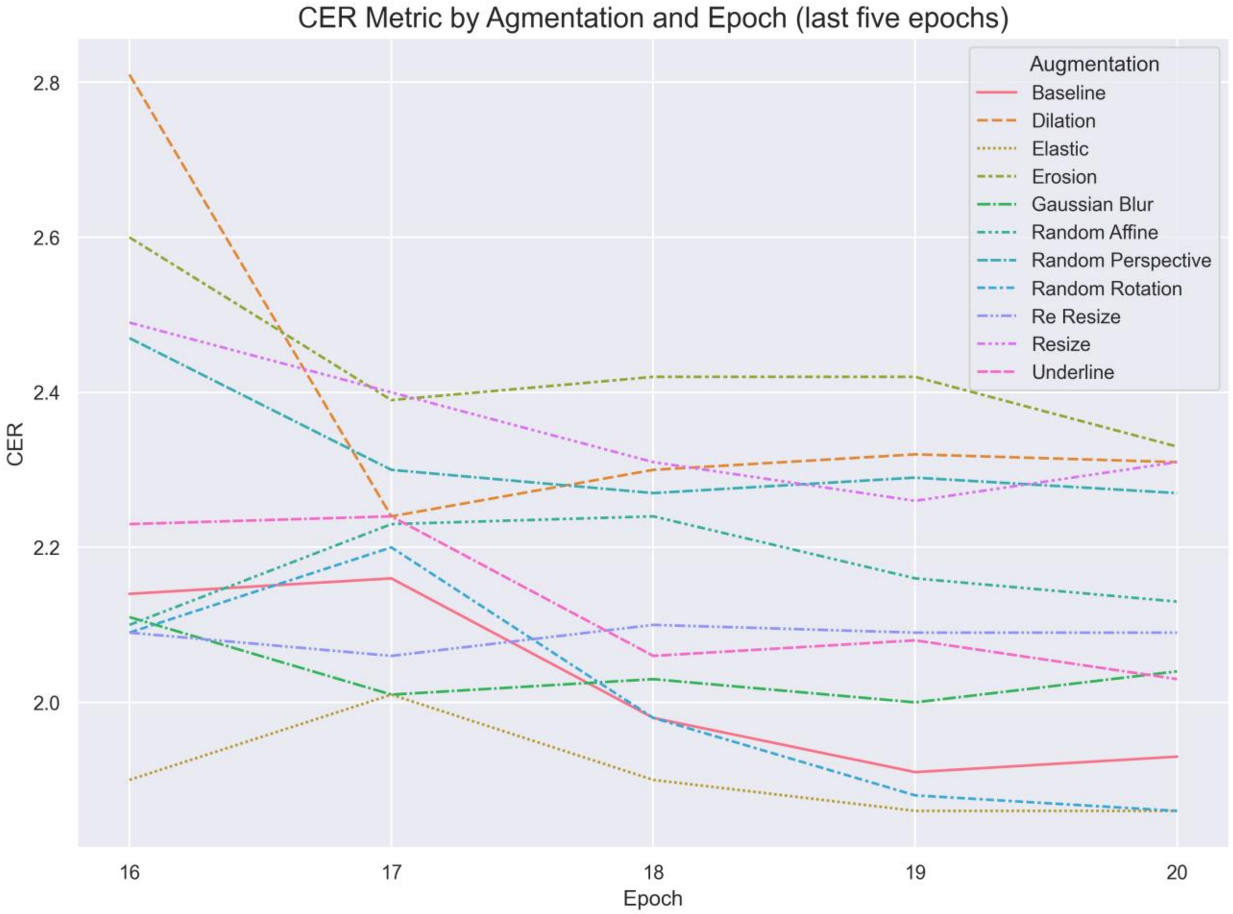


Figure 5.5 The performance of CER in the last five epochs. The training is close to converging with clear, better augmentations.

5.4 Character Analysis

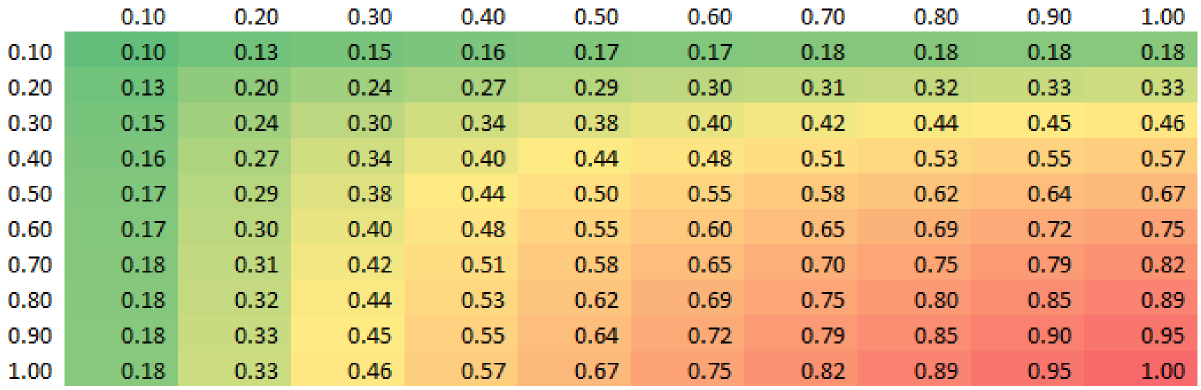
Another helpful way to evaluate the efficiency of the models is by other metrics. In this section, we will discuss three different metrics: precision (5.1), recall (5.2), and f1 (5.3). In Handwritten Text Recognition, precision measures the confidence that a prediction is correct and how many characters are predicted correctly. Recall measures the confidence that prediction is complete; how many characters were predicted correctly. When using them alone, these two measures have significant disadvantages in HTR. We might get a predictor with high precision in predicting a specific character but a high false negative, e.g., predicting incorrectly this particular character. We might also have high recall and get a predictor with high false positives where it falsely predicts many characters, as is the specific character.

One solution to the disadvantages of precision and recall is the F-score family, commonly used as the f1 score (5.3). F1 is the harmonic mean to two values, in our case, precision and recall. As seen in Figure 5.6, a higher score of f1 is highly correlated to higher scores in precision and recall. To our problem (HTR), a predictor with f1=1 means that its predictions are correct *and* complete.

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{5.1}$$

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{5.2}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.3}$$



	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00
0.10	0.10	0.13	0.15	0.16	0.17	0.17	0.18	0.18	0.18	0.18
0.20	0.13	0.20	0.24	0.27	0.29	0.30	0.31	0.32	0.33	0.33
0.30	0.15	0.24	0.30	0.34	0.38	0.40	0.42	0.44	0.45	0.46
0.40	0.16	0.27	0.34	0.40	0.44	0.48	0.51	0.53	0.55	0.57
0.50	0.17	0.29	0.38	0.44	0.50	0.55	0.58	0.62	0.64	0.67
0.60	0.17	0.30	0.40	0.48	0.55	0.60	0.65	0.69	0.72	0.75
0.70	0.18	0.31	0.42	0.51	0.58	0.65	0.70	0.75	0.79	0.82
0.80	0.18	0.32	0.44	0.53	0.62	0.69	0.75	0.80	0.85	0.89
0.90	0.18	0.33	0.45	0.55	0.64	0.72	0.79	0.85	0.90	0.95
1.00	0.18	0.33	0.46	0.57	0.67	0.75	0.82	0.89	0.95	1.00

Figure 5.6 F1 is the combination of precision and recall. Higher values of f1 IFF precision and recall are high. This figure demonstrates the correlation between these three matrices.

Table 5.2 F1 score for the small-case characters. Other characters with low support are not included for visibility. Macro average is the unweighted average of all f1 scores. The weighted average uses the support to calculate the average. Accuracy is general accuracy. F1 scale up to 100 to improve the visibility.

	Baseline	Random Rotation	Gaussian Blur	Dilation	Erosion	Resize	Underline	Random Affine	Random Perspective	Elastic	Re Resize
space	99.51	99.57	99.40	99.53	99.34	99.42	99.47	99.53	99.55	99.59	99.57
a	99.28	99.20	98.95	98.87	98.86	98.99	99.16	98.91	98.48	99.16	98.69
b	97.12	98.41	97.79	97.14	97.79	97.78	97.78	99.05	96.53	97.47	96.53
c	98.96	99.12	99.20	98.80	98.65	98.96	99.28	99.04	98.48	99.44	99.12
d	98.60	99.11	98.98	98.98	98.61	98.86	99.49	98.73	98.48	99.49	99.24
e	98.59	98.71	98.53	98.15	97.99	98.17	98.29	98.41	98.16	98.41	98.29
f	98.32	98.05	97.78	98.59	98.06	97.51	98.31	98.33	98.05	97.77	97.79
g	98.68	99.33	99.01	99.34	99.01	98.68	98.68	99.01	98.68	99.67	98.68
h	98.21	98.21	98.21	97.51	96.77	98.21	97.49	98.19	97.12	98.56	97.08
i	98.98	99.39	99.18	98.95	98.84	99.11	99.15	98.81	98.67	98.84	99.15
l	99.32	99.22	99.13	98.93	99.22	98.92	99.12	99.03	98.53	99.12	99.12
m	98.51	98.30	98.31	98.24	98.17	98.10	98.31	97.91	98.03	98.31	98.31
n	98.11	98.23	98.16	97.76	98.28	98.05	97.99	98.11	98.17	98.50	97.87
o	98.71	98.60	98.25	97.43	98.13	98.08	98.43	97.79	97.73	98.26	98.02
p	99.20	99.31	98.97	98.17	99.43	98.52	99.08	98.86	99.08	99.54	99.08
q	98.84	98.84	98.27	97.39	98.27	97.42	97.97	98.56	98.84	98.55	96.83
r	98.69	98.34	98.50	98.29	98.15	97.94	98.14	98.19	98.59	98.75	98.64
s	99.08	99.03	98.94	98.89	98.80	98.61	98.85	98.94	98.71	98.98	98.85
t	99.27	99.35	98.94	98.90	99.19	99.23	99.23	99.47	99.03	99.15	99.27
u	98.41	98.51	98.64	98.00	98.38	98.14	98.28	98.60	98.51	98.73	98.33
v	96.36	97.48	97.94	97.73	96.80	97.01	97.51	96.82	97.26	97.27	96.77
x	97.84	97.18	98.59	97.87	96.50	96.45	97.18	97.14	95.65	98.59	97.84
y	100.00	100.00	100.00	90.91	100.00	95.24	100.00	100.00	100.00	100.00	100.00
accuracy	98.08	98.13	98.00	97.75	97.68	97.74	97.96	97.90	97.73	98.14	97.93
macro avg	80.79	83.36	78.34	80.61	81.52	84.22	78.08	83.66	82.39	82.87	82.49
weighted avg	98.26	98.26	98.09	97.90	97.61	97.90	98.03	97.90	97.81	98.25	98.03

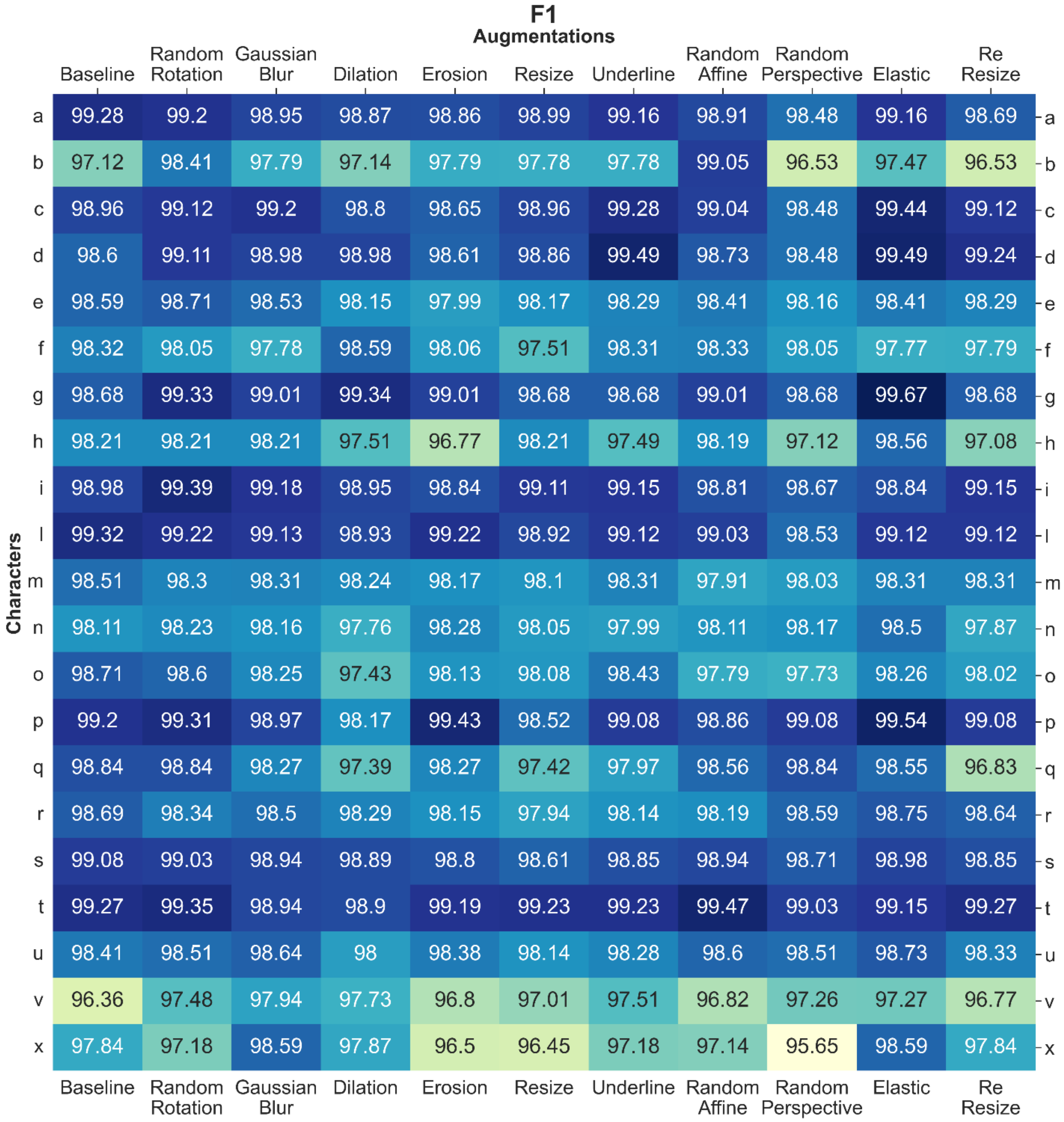
Table 5.2 shows the f1 scores for each lowercase character and each augmentation. Another way to see the f1 score performance is from Figure 5.7. We can see that some characters are easier to predict, e.g., ‘i’ and ‘l,’ while others are hard to predict. Another important finding,

besides the fact that, as discussed, some models are better than others, is that some weak models have high f1 scores in specific characters underline and the character ‘d’.

The confusion matrix in Figure 5.8 shows that average models have confusion between some characters, e.g., ‘m’ and ‘n’. Further looking into each specific model (not supplied here) shows that different models have slightly different confusion matrices. These findings we discussed in the past paragraphs suggest that although augmentation models can be ranked by their general accuracy (CER), they have different efficiencies. These lead us to the following discussion: can we improve the overall performance by prediction by combining all or part of the augmentation models?

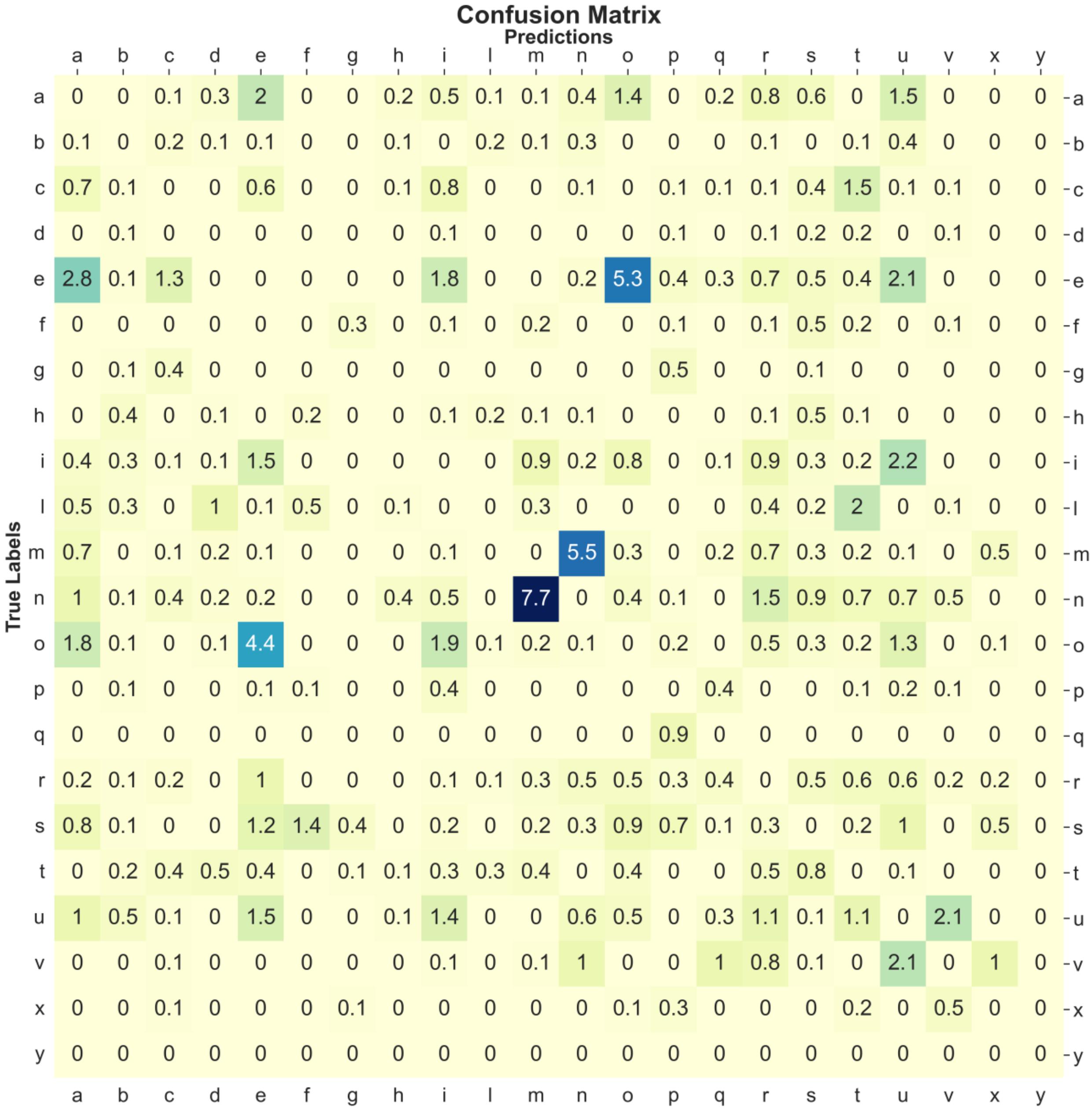
5.5 Ensemble Learning

Ensemble learning is learning from multiple algorithms, in our case, multiple augmentation models. There are multiple ways to compose different models into a single predictor. In our work, we use the Voting form of ensemble. We did it in two ways: A: for each predicted sentence, we output the sentence that got the most votes from all eleven predictors. B: the voting was only between the best five models with the best overall f1 score (Elastic, Random Rotation, Underline, Gaussian Blur, and Baseline. These two ways look at the predicted sentence; we tried another more elaborate way by voting on each character. Although promising, this third way suffers from instability and was not included in our research. The average CER of all models is 2.11 in the range of 1.86 to 2.33. Ensemble A has a CER of 1.66, and Ensemble B has a CER of 1.60.



Characters	Baseline	Random Rotation	Gaussian Blur	Dilation	Erosion	Resize	Underline	Random Affine	Random Perspective	Elastic	Re Resize
a	99.28	99.2	98.95	98.87	98.86	98.99	99.16	98.91	98.48	99.16	98.69
b	97.12	98.41	97.79	97.14	97.79	97.78	97.78	99.05	96.53	97.47	96.53
c	98.96	99.12	99.2	98.8	98.65	98.96	99.28	99.04	98.48	99.44	99.12
d	98.6	99.11	98.98	98.98	98.61	98.86	99.49	98.73	98.48	99.49	99.24
e	98.59	98.71	98.53	98.15	97.99	98.17	98.29	98.41	98.16	98.41	98.29
f	98.32	98.05	97.78	98.59	98.06	97.51	98.31	98.33	98.05	97.77	97.79
g	98.68	99.33	99.01	99.34	99.01	98.68	98.68	99.01	98.68	99.67	98.68
h	98.21	98.21	98.21	97.51	96.77	98.21	97.49	98.19	97.12	98.56	97.08
i	98.98	99.39	99.18	98.95	98.84	99.11	99.15	98.81	98.67	98.84	99.15
l	99.32	99.22	99.13	98.93	99.22	98.92	99.12	99.03	98.53	99.12	99.12
m	98.51	98.3	98.31	98.24	98.17	98.1	98.31	97.91	98.03	98.31	98.31
n	98.11	98.23	98.16	97.76	98.28	98.05	97.99	98.11	98.17	98.5	97.87
o	98.71	98.6	98.25	97.43	98.13	98.08	98.43	97.79	97.73	98.26	98.02
p	99.2	99.31	98.97	98.17	99.43	98.52	99.08	98.86	99.08	99.54	99.08
q	98.84	98.84	98.27	97.39	98.27	97.42	97.97	98.56	98.84	98.55	96.83
r	98.69	98.34	98.5	98.29	98.15	97.94	98.14	98.19	98.59	98.75	98.64
s	99.08	99.03	98.94	98.89	98.8	98.61	98.85	98.94	98.71	98.98	98.85
t	99.27	99.35	98.94	98.9	99.19	99.23	99.23	99.47	99.03	99.15	99.27
u	98.41	98.51	98.64	98	98.38	98.14	98.28	98.6	98.51	98.73	98.33
v	96.36	97.48	97.94	97.73	96.8	97.01	97.51	96.82	97.26	97.27	96.77
x	97.84	97.18	98.59	97.87	96.5	96.45	97.18	97.14	95.65	98.59	97.84

Figure 5.7 F1 (normalized to 100) for augmentations and small case characters.



	a	b	c	d	e	f	g	h	i	l	m	n	o	p	q	r	s	t	u	v	x	y	
a	0	0	0.1	0.3	2	0	0	0.2	0.5	0.1	0.1	0.4	1.4	0	0.2	0.8	0.6	0	1.5	0	0	0	a
b	0.1	0	0.2	0.1	0.1	0	0	0.1	0	0.2	0.1	0.3	0	0	0	0.1	0	0.1	0.4	0	0	0	b
c	0.7	0.1	0	0	0.6	0	0	0.1	0.8	0	0	0.1	0	0.1	0.1	0.1	0.4	1.5	0.1	0.1	0	0	c
d	0	0.1	0	0	0	0	0	0	0.1	0	0	0	0	0.1	0	0.1	0.2	0.2	0	0.1	0	0	d
e	2.8	0.1	1.3	0	0	0	0	0	1.8	0	0	0.2	5.3	0.4	0.3	0.7	0.5	0.4	2.1	0	0	0	e
f	0	0	0	0	0	0	0.3	0	0.1	0	0.2	0	0	0.1	0	0.1	0.5	0.2	0	0.1	0	0	f
g	0	0.1	0.4	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0.1	0	0	0	0	0	g
h	0	0.4	0	0.1	0	0.2	0	0	0.1	0.2	0.1	0.1	0	0	0	0.1	0.5	0.1	0	0	0	0	h
i	0.4	0.3	0.1	0.1	1.5	0	0	0	0	0	0.9	0.2	0.8	0	0.1	0.9	0.3	0.2	2.2	0	0	0	i
l	0.5	0.3	0	1	0.1	0.5	0	0.1	0	0	0.3	0	0	0	0	0.4	0.2	2	0	0.1	0	0	l
m	0.7	0	0.1	0.2	0.1	0	0	0	0.1	0	0	5.5	0.3	0	0.2	0.7	0.3	0.2	0.1	0	0.5	0	m
n	1	0.1	0.4	0.2	0.2	0	0	0.4	0.5	0	7.7	0	0.4	0.1	0	1.5	0.9	0.7	0.7	0.5	0	0	n
o	1.8	0.1	0	0.1	4.4	0	0	0	1.9	0.1	0.2	0.1	0	0.2	0	0.5	0.3	0.2	1.3	0	0.1	0	o
p	0	0.1	0	0	0.1	0.1	0	0	0.4	0	0	0	0	0	0.4	0	0	0.1	0.2	0.1	0	0	p
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	q
r	0.2	0.1	0.2	0	1	0	0	0	0.1	0.1	0.3	0.5	0.5	0.3	0.4	0	0.5	0.6	0.6	0.2	0.2	0	r
s	0.8	0.1	0	0	1.2	1.4	0.4	0	0.2	0	0.2	0.3	0.9	0.7	0.1	0.3	0	0.2	1	0	0.5	0	s
t	0	0.2	0.4	0.5	0.4	0	0.1	0.1	0.3	0.3	0.4	0	0.4	0	0	0.5	0.8	0	0.1	0	0	0	t
u	1	0.5	0.1	0	1.5	0	0	0.1	1.4	0	0	0.6	0.5	0	0.3	1.1	0.1	1.1	0	2.1	0	0	u
v	0	0	0.1	0	0	0	0	0	0.1	0	0.1	1	0	0	1	0.8	0.1	0	2.1	0	1	0	v
x	0	0	0.1	0	0	0	0.1	0	0	0	0	0	0.1	0.3	0	0	0	0.2	0	0.5	0	0	x
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	y
	a	b	c	d	e	f	g	h	i	l	m	n	o	p	q	r	s	t	u	v	x	y	

Figure 5.8 Confusion matrix between true and predicted labels. The values are the average between all eleven models.

5.6 Comparison to other papers

[28] in their work on the Gwalther dataset and TrOCR, they achieved the best CER score of 3.18 in the tenth epoch. They also gave a SOTA model HTR+ to achieve a CER of 2.74 in the 50th epoch. As presented previously, all our models achieved better scores. The differences

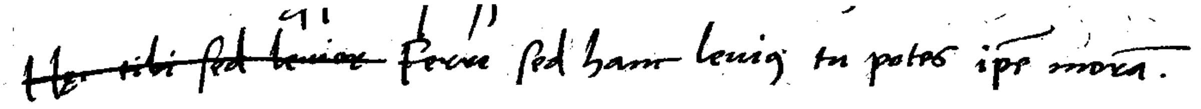
between our scores and [28] require explanation. We suggest that since TrOCR was pre-trained on single-line images. The original Gwalther dataset has many images that include two or more lines. We think that [28] used the dataset as is, while we improved the dataset's quality by cleaning all the images to be friendly to TrOCR by making all the images a single line.

5.7 Examples

Here, we will give a few examples where the models have challenges to recognize.

5.7.1 First Example

In this example, we will examine line 22 from file 1111690 Figure 5.9 and see how different augmentation models predict the line in Table 5.3. This line has a deletion by overline mark and line overflow from the top line.



~~He tibi sed levior~~ Ferre sed hanc levius tu potes ipse moram.

Figure 5.9 Line 22 from file 1111690

Table 5.3 Models and their predictions to line 22 of 1111690. The last line is the label.

Model	Prediction
Baseline	Hei sed ferre sed hanc levig tu potes ipse moram.
Random Rotation	Heu ferre sed hanc levis tu potes ipse moram.
Gaussian Blur	He lectled ferre sed hanc levis tu potes ipse moram.
Dilation	He tibi servi sed hanc levius tu potes ipse moram.
Erosion	Hęc tibi ded levior ferre sed hanc levis tu potes ipse moram.
Resize	Hei sed ferre sed hanc levique tu potes ipse moram.
Underline	He led ferne ferre sed hanc levis tu potes ipse moram.
Random Affine	Perre sed hanc levis tu potes ipse moram.
Random Perspective	Hei ferre sed hanc levius tu potes ipse moram.
Elastic	He ferre, ferre sed hanc levique tu potes ipse moram.
Re Resize	He deced ferre sed hanc levique tu potes ipse moram.
Label	Ferre sed hanc levius tu potes ipse moram.

5.7.2 Second Example

In this example, we will examine line 15 from file 1111823 Figure 5.10 and see how different augmentation models predict the line in Table 5.4. This first word in this line has excessive drawing, and the second to last word has deletion by scribble.

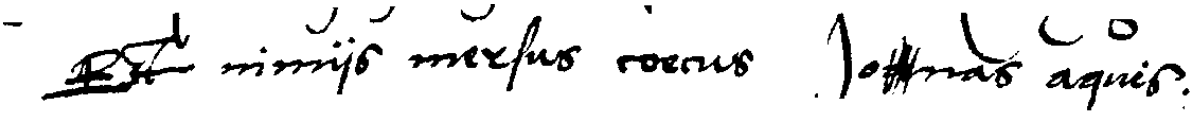


Figure 5.10 Line 15 from file 1111823

Table 5.4 Models and their predictions to line 15 of 1111823. The last line is the label.

Model	**Prediction**
Baseline	Et nimiis mersus coecus vernas aquis.
Random Rotation	Et nimiis mersus coecus somnas aquis.
Gaussian Blur	Et nimiis mersus coecus formas aquis.
Dilation	Et nimiis mersus coecus Iothas aquis.
Erosion	Et nimiis mersus coecus formas aquis.
Resize	Et nimiis mersus coecus fortnas aquis.
Underline	Et nimiis mersus coecus vernas aquis.
Random Affine	Et nimiis mersus coecus poenas aquis.
Random Perspective	Et nimiis mersus coecus poenas aquis.
Elastic	Et nimis mersus coecus Iottinas aquis.
Re Resize	Est nimiis mersus coecus formas aquis.
Label	Est nimiis mersus coecus Ionas aquis.

5.7.3 Third Example

In this example, we will examine line 0 from file 1111832 Figure 5.11 and see how different augmentation models predict the line in Table 5.5. This line has background noise, probably because it is close to the edge of the page and prunes to an aging effect. Also, the writing is blurred.

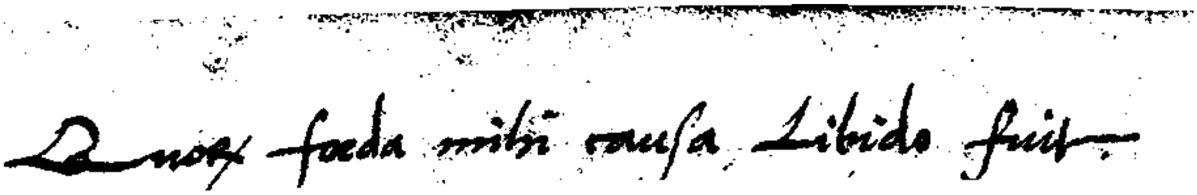

Quorū foeda mihi causa Libido fuit.

Figure 5.11 Line 0 from file 1111832

Table 5.5 Models and their predictions to line 0 of 1111832. The last line is the label.

Model	Prediction
Baseline	Quorum foedera mihi causa libido fuit.
Random Rotation	Quorum foedera mihi causa libido fuit.
Gaussian Blur	Quorum foedera mihi causa libido fuit.
Dilation	Quorum fada mihi causa libido fuit.
Erosion	Quorum foeda mihi causa libido fuit.
Resize	Quorum foedera mihi causa libido fuit.
Underline	Quorum foedera mihi causa libido fuit.
Random Affine	Quorum foedera mihi causa libido fuit.
Random Perspective	Quorum foedera mihi causa libido fuit.
Elastic	Quorum foeder mihi causa libido fuit.
Re Resize	Quorum foeda mihi causa libido fuit.
Label	Quorum foeda mihi causa Libido fuit.

CHAPTER VI

DISCUSSION

6.1 Introduction

We have seen in the previous chapters that TrOCR can achieve state-of-the-art HTR of historical manuscripts. Our work, which follows the dataset used [28], performed much better than the fine-tuned TrOCR$_{\mathrm{BASE}}$ and SOTA HTR+ in the same paper. The building blocks of our model are well-prepared line images, fine-tuning TrOCR, augmentations, and ensemble learning. In this chapter, we will discuss these parts of our solution.

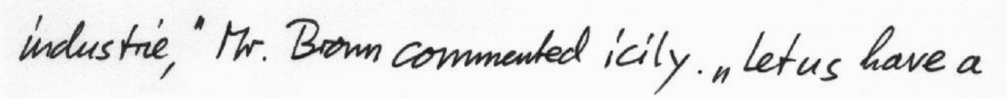
6.2 Preprocessing and fine-tuning for TrOCR

In our work, we fine-tuned TrOCR to recognize images it did not see. Selecting the correct hyperparameter in fine-tuning can improve the performance of the final model [35]. However, choosing the correct hyperparameters can be challenging and requires an extensive search [35], [36]. To avoid this, we prepared the line images to be similar to those originally TrOCR was trained on and used similar hyperparameters used in the pre-training of TrOCR [36].

Handwritten TrOCR was pre-trained with line images from the IAM dataset[6], see Figure 6.1. TrOCR expects line images with a single line. However, in our dataset, lines' coordinates in XML files (**Error! Reference source not found.**) include points overflowing to neighbor lines, s o the output images have more than one line. Although some other algorithms can handle this

[6] https://fki.tic.heia-fr.ch/databases/iam-handwriting-database

situation, this negatively affects the performance of TrOCR. We corrected the coordinates so line images have only one line of text. Also, we converted the images to black and white images. These two steps make our images very similar to the images TrOCR was pre-trained on, enabling us to choose hyperparameters close to the pre-training step of TrOCR, eventually improving the performance.



industrie," Mr. Brown commented icily. „ letus have a

Figure 6.1 IAM image

6.3 Augmentation

Image augmentation is a well-known tool to improve the performance of models [22], [34], [37]. Our findings also found that some augmentations improved the TrOCR performance while others did not. However, a deep investigation of the augmentation results shows that different models perform differently on the overall and specific character bases. This diversity [38] leads us naturally to seek an ensemble learning model.

6.4 Ensemble learning

Ensemble learning [39]–[42] is an efficient algorithm to join similar but different learners to achieve better performance than any learners alone. Our work also shows that the best augmentation–the elastic model has a CER score of 1.86. When we ensemble all the algorithms for a vote, we achieve a CER of 1.66, and if we take a vote of the five models with the best F1, we even get a better CER of 1.60.

CHAPTER VII

CONCLUSION

In this work, we presented a SOTA model to recognize Latin text from 16th-century manuscripts. We showed that with carefully prepared images, transformer-based algorithms, augmentations, and ensemble learning, it is possible to recognize historical manuscripts with a Character Error Rate (CER) of 1.60, an improvement of 50% from the previous $\text{TrOCR}_{\text{BASE}}$-based solution and a gain of 42% from other previous SOTA solution [28].

TrOCR constitutes two pluggable pre-trained transformers: an encoder and a decoder. The former is a vision transformer that encodes line images into a “code” consumed by the latter language transformer, which finally outputs the predicted text. Originally, TrOCR was pre-trained in a single language, English. In our work, which followed [28], we showed that plugging a multilanguage transformer decoder (XLM-RoBERTa) extends TrOCR into a multilanguage text recognizer. We used SOTA tools, Python, PyTorch, and Huggingface to achieve this, which already implemented this extension to TrOCR.

Further research will extend our work to other languages with different character sets, e.g., Hebrew and Arabic. Another possible extension to our work is character-based ensemble learning, where voting is done on the character level and not on the line level, as done in this work.

REFERENCES

[1] L. J. Cappon, “Historical Manuscripts as Archives: Some Definitions and Their Application,” *The American Archivist*, vol. 19, no. 2, pp. 101–110, 1956.

[2] J. Makhoul, R. Schwartz, C. Lapre, and I. Bazzi, “A script-independent methodology for optical character recognition,” *Pattern Recognition*, vol. 31, no. 9, pp. 1285–1294, 1998.

[3] L. Wolf, L. Potikha, N. Dershowitz, R. Shweka, and Y. Choueka, “Computerized paleography: Tools for historical manuscripts,” in *2011 18th IEEE International Conference on Image Processing*, Brussels, Belgium: IEEE, Sep. 2011, pp. 3545–3548. doi: 10.1109/ICIP.2011.6116481.

[4] T. M. Rath and R. Manmatha, “Features for word spotting in historical manuscripts,” in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, Edinburgh, UK: IEEE Comput. Soc, 2003, pp. 218–222. doi: 10.1109/ICDAR.2003.1227662.

[5] J. A. Rodríguez-Serrano and F. Perronnin, “A Model-Based Sequence Similarity with Application to Handwritten Word Spotting,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2108–2120, Nov. 2012, doi: 10.1109/TPAMI.2012.25.

[6] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke, “A Novel Word Spotting Method Based on Recurrent Neural Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 2, pp. 211–224, Feb. 2012, doi: 10.1109/TPAMI.2011.113.

[7] K. Dutta, P. Krishnan, M. Mathew, and C. V. Jawahar, “Improving CNN-RNN Hybrid Networks for Handwriting Recognition,” in *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, Aug. 2018, pp. 80–85. doi: 10.1109/ICFHR-2018.2018.00023.

[8] A. Vaswani *et al.*, “Attention Is All You Need.” arXiv, Dec. 05, 2017. doi: 10.48550/arXiv.1706.03762.

[9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” arXiv, May 24, 2019. doi: 10.48550/arXiv.1810.04805.

[10] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.” arXiv, Jun. 03, 2021. doi: 10.48550/arXiv.2010.11929.

[11] M. Li *et al.*, “TrOCR: Transformer-based Optical Character Recognition with Pre-trained Models.” arXiv, Sep. 06, 2022. doi: 10.48550/arXiv.2109.10282.

[12] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate.” arXiv, May 19, 2016. doi: 10.48550/arXiv.1409.0473.

[13] K. Cho, B. Van Merrienboer, D. Bahdanau, and Y. Bengio, “On the Properties of Neural Machine Translation: Encoder–Decoder Approaches,” in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, Doha, Qatar: Association for Computational Linguistics, 2014, pp. 103–111. doi: 10.3115/v1/W14-4012.

[14] D. Kass and E. Vats, “AttentionHTR: Handwritten Text Recognition Based on Attention Encoder-Decoder Networks.” arXiv, Sep. 12, 2022. Accessed: Jul. 20, 2023. [Online]. Available: http://arxiv.org/abs/2201.09390

[15] J. Michael, R. Labahn, T. Grüning, and J. Zöllner, “Evaluating Sequence-to-Sequence Models for Handwritten Text Recognition.” arXiv, Jul. 15, 2019. doi: 10.48550/arXiv.1903.07377.

[16] Y. Li, K. Zhang, J. Cao, R. Timofte, and L. Van Gool, “LocalViT: Bringing Locality to Vision Transformers.” arXiv, Apr. 12, 2021. Accessed: Jul. 26, 2023. [Online]. Available: http://arxiv.org/abs/2104.05707

[17] T. Wolf *et al.*, “HuggingFace’s Transformers: State-of-the-art Natural Language Processing.” arXiv, Jul. 13, 2020. doi: 10.48550/arXiv.1910.03771.

[18] A. Conneau *et al.*, “Unsupervised Cross-lingual Representation Learning at Scale.” arXiv, Apr. 07, 2020. doi: 10.48550/arXiv.1911.02116.

[19] Y. Liu *et al.*, “RoBERTa: A Robustly Optimized BERT Pretraining Approach.” arXiv, Jul. 26, 2019. doi: 10.48550/arXiv.1907.11692.

[20] J. Han *et al.*, “You Only Cut Once: Boosting Data Augmentation with a Single Cut,” in *Proceedings of the 39th International Conference on Machine Learning*, PMLR, Jun. 2022, pp. 8196–8212. Accessed: Jul. 26, 2023. [Online]. Available: https://proceedings.mlr.press/v162/han22a.html

[21] Ms. A. Bansal, Dr. R. Sharma, and Dr. M. Kathuria, “A Systematic Review on Data Scarcity Problem in Deep Learning: Solution and Applications,” *ACM Comput. Surv.*, vol. 54, no. 10s, p. 208:1-208:29, Sep. 2022, doi: 10.1145/3502287.

[22] S. Minz, R. Kanojia, T. Yadav, and N. Jayanthi, “Enhancing Accuracy in Handwritten Text Recognition with Convolutional Recurrent Neural Network and Data Augmentation Techniques,” in *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*, May 2023, pp. 803–808. doi: 10.1109/ICSCCC58608.2023.10176601.

[23] J. Puigcerver, “Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Kyoto: IEEE, Nov. 2017, pp. 67–72. doi: 10.1109/ICDAR.2017.20.

[24] A. F. de Sousa Neto, B. Leite Dantas Bezerra, A. Hector Toselli, and E. Baptista Lima, “A robust handwritten recognition system for learning on different data restriction scenarios,” *Pattern Recognition Letters*, vol. 159, pp. 232–238, Jul. 2022, doi: 10.1016/j.patrec.2022.04.009.

[25] T. Wilkinson and A. Brun, “Semantic and Verbatim Word Spotting Using Deep Neural Networks,” in *15th International Conference on Frontiers in Handwriting Recognition, ICFHR 2016, Shenzhen, China, October 23-26, 2016*, IEEE Computer Society, 2016, pp. 307–312. doi: 10.1109/ICFHR.2016.0065.

[26] G. Retsinas, G. Sfikas, B. Gatos, and C. Nikou, “Best Practices for a Handwritten Text Recognition System,” in *Document Analysis Systems*, S. Uchida, E. Barney, and V. Eglin, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 247–259. doi: 10.1007/978-3-031-06555-2_17.

[27] J. A. Sánchez, V. Romero, A. H. Toselli, M. Villegas, and E. Vidal, “A set of benchmarks for Handwritten Text Recognition on historical documents,” *Pattern Recognition*, vol. 94, pp. 122–134, Oct. 2019, doi: 10.1016/j.patcog.2019.05.025.

[28] P. B. Ströbel, S. Clematide, M. Volk, and T. Hodel, “Transformer-based HTR for Historical Documents.” arXiv, Mar. 21, 2022. Accessed: Jul. 20, 2023. [Online]. Available: http://arxiv.org/abs/2203.11008

[29] E. Chammas, C. Mokbel, and L. Likforman-Sulem, “Handwriting Recognition of Historical Documents with few labeled data,” in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, Apr. 2018, pp. 43–48. doi: 10.1109/DAS.2018.15.

[30] E. S. M. Penn, “Exploring archival value: an axiological approach,” PhD Thesis, UCL (University College London), 2014.

[31] P. Stotz and P. Ströbel, “bullinger-digital/gwalther-handwriting-ground-truth: Initial release.” Zenodo, May 22, 2021. doi: 10.5281/zenodo.4780947.

[32] L. Seaward and M. Kallio, “Transkribus: Handwritten Text Recognition technology for historical documents.,” in *DH*, 2017.

[33] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *MACHINE LEARNING IN PYTHON*.

[34] M. Xu, S. Yoon, A. Fuentes, and D. S. Park, “A Comprehensive Survey of Image Augmentation Techniques for Deep Learning,” *Pattern Recognition*, vol. 137, p. 109347, May 2023, doi: 10.1016/j.patcog.2023.109347.

[35] M. Subramanian, K. Shanmugavadivel, and P. S. Nandhini, “On fine-tuning deep learning models using transfer learning and hyper-parameters optimization for disease identification in maize leaves,” *Neural Comput & Applic*, vol. 34, no. 16, pp. 13951–13968, Aug. 2022, doi: 10.1007/s00521-022-07246-w.

[36] H. Li *et al.*, “Rethinking the Hyperparameters for Fine-tuning.” arXiv, Feb. 19, 2020. doi: 10.48550/arXiv.2002.11770.

[37] R. Heil and E. Breznik, “A Study of Augmentation Methods for Handwritten Stenography Recognition.” arXiv, Mar. 05, 2023. Accessed: Jul. 27, 2023. [Online]. Available: http://arxiv.org/abs/2303.02761

[38] L. I. Kuncheva and C. J. Whitaker, “Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy,” *Machine Learning*, vol. 51, no. 2, pp. 181–207, May 2003, doi: 10.1023/A:1022859003006.

[39] D. Opitz and R. Maclin, “Popular Ensemble Methods: An Empirical Study,” *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, Aug. 1999, doi: 10.1613/jair.614.

[40] R. Polikar, “Ensemble based systems in decision making,” *IEEE Circuits and Systems Magazine*, vol. 6, no. 3, pp. 21–45, 2006, doi: 10.1109/MCAS.2006.1688199.

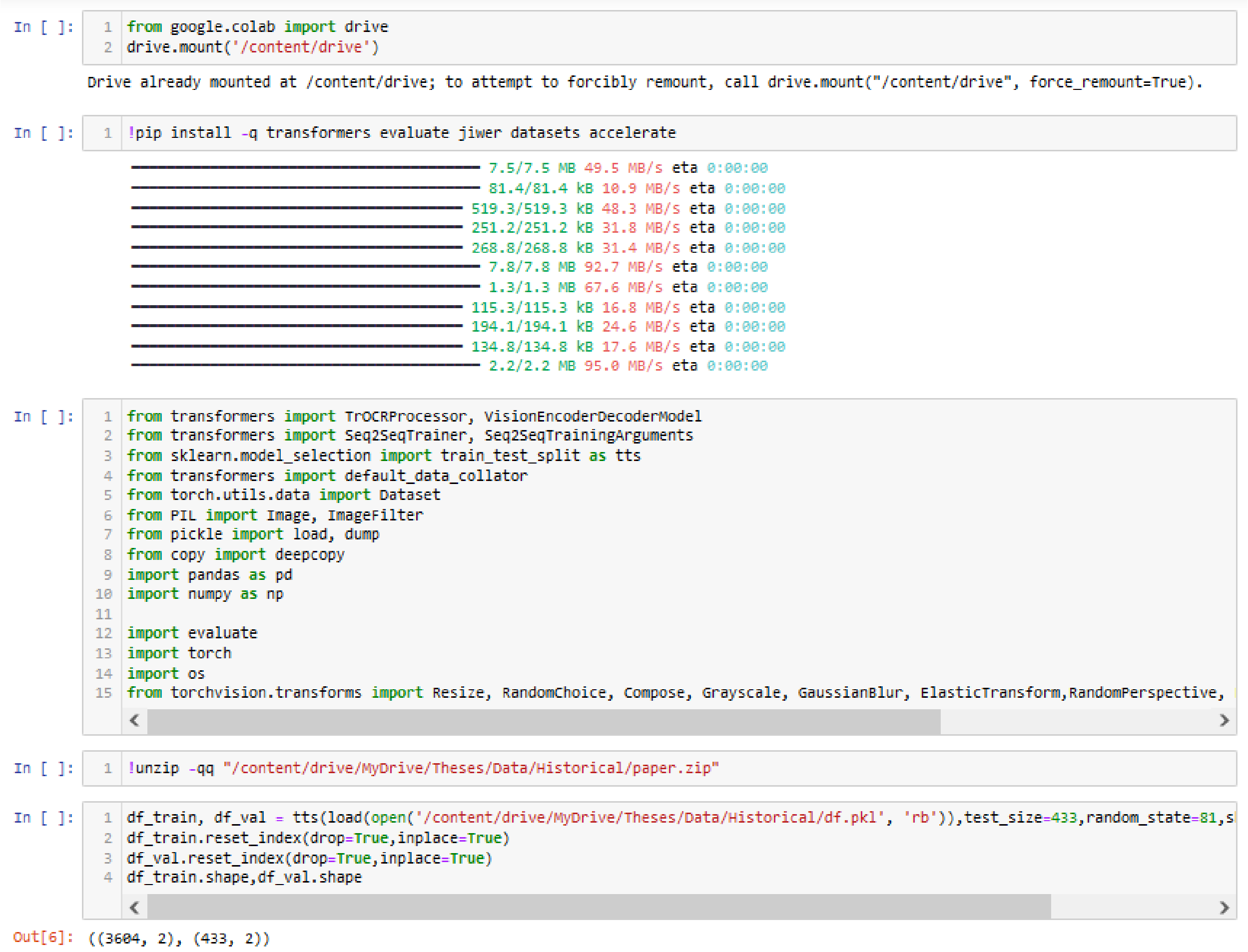
[41] L. Rokach, “Ensemble-based classifiers,” *Artif Intell Rev*, vol. 33, no. 1, pp. 1–39, Feb. 2010, doi: 10.1007/s10462-009-9124-7.

[42] I. D. Mienye and Y. Sun, “A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects,” *IEEE Access*, vol. 10, pp. 99129–99149, 2022, doi: 10.1109/ACCESS.2022.3207287.

APPENDIX A

TRAINING AND VALIDATION CODE

This appendix includes the training and validation code we used in our work. We used this code on Google Colab. Part of the code is from TrOCR's code[7], Huggingface[8] , and its referenced tutorial notebooks.[9]



```
In [ ]: from google.colab import drive
        drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
In [ ]: !pip install -q transformers evaluate jiwer datasets accelerate

     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.5/7.5 MB 49.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.4/81.4 kB 10.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 519.3/519.3 kB 48.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 251.2/251.2 kB 31.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 268.8/268.8 kB 31.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 92.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 67.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.3/115.3 kB 16.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.1/194.1 kB 24.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 17.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 95.0 MB/s eta 0:00:00
```

```
In [ ]: from transformers import TrOCRProcessor, VisionEncoderDecoderModel
        from transformers import Seq2SeqTrainer, Seq2SeqTrainingArguments
        from sklearn.model_selection import train_test_split as tts
        from transformers import default_data_collator
        from torch.utils.data import Dataset
        from PIL import Image, ImageFilter
        from pickle import load, dump
        from copy import deepcopy
        import pandas as pd
        import numpy as np

        import evaluate
        import torch
        import os
        from torchvision.transforms import Resize, RandomChoice, Compose, Grayscale, GaussianBlur, ElasticTransform,RandomPerspective,
```

```
In [ ]: !unzip -qq "/content/drive/MyDrive/Theses/Data/Historical/paper.zip"
```

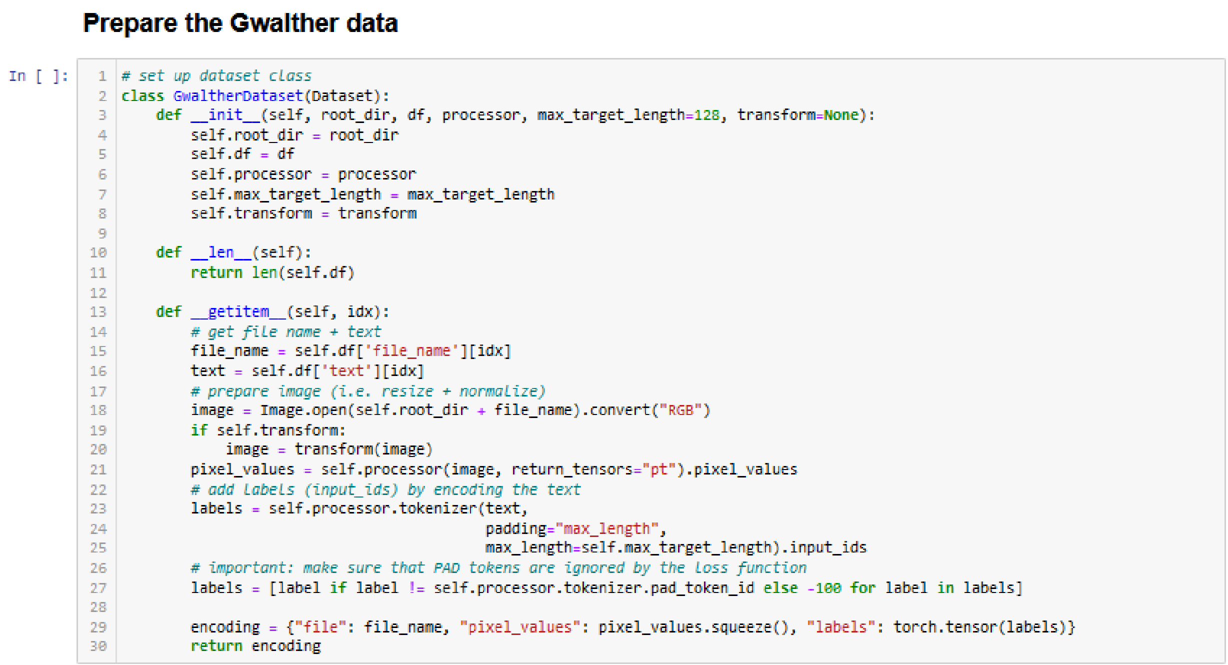
```
In [ ]: df_train, df_val = tts(load(open('/content/drive/MyDrive/Theses/Data/Historical/df.pkl', 'rb')),test_size=433,random_state=81,s
        df_train.reset_index(drop=True,inplace=True)
        df_val.reset_index(drop=True,inplace=True)
        df_train.shape,df_val.shape

Out[6]: ((3604, 2), (433, 2))
```

Figure A.1 Preparing the Colab environment and importing required libraries

[7] https://github.com/microsoft/unilm/tree/master/trocr

[8] https://huggingface.co/docs/transformers/model_doc/trocr

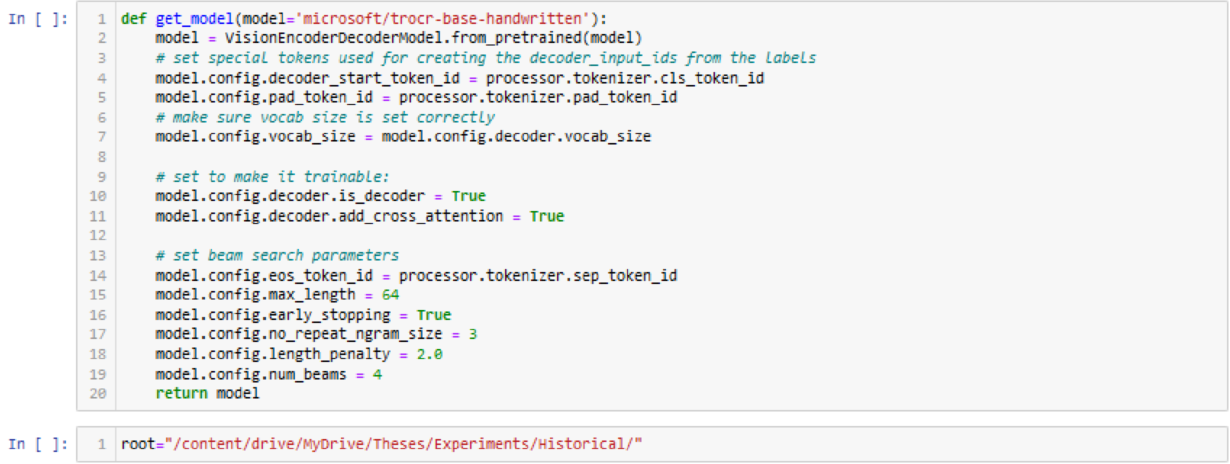
[9] https://github.com/NielsRogge/Transformers-Tutorials/tree/master/TrOCR



Prepare the Gwalther data

```python
# set up dataset class
class GwaltherDataset(Dataset):
    def __init__(self, root_dir, df, processor, max_target_length=128, transform=None):
        self.root_dir = root_dir
        self.df = df
        self.processor = processor
        self.max_target_length = max_target_length
        self.transform = transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        # get file name + text
        file_name = self.df['file_name'][idx]
        text = self.df['text'][idx]
        # prepare image (i.e. resize + normalize)
        image = Image.open(self.root_dir + file_name).convert("RGB")
        if self.transform:
            image = transform(image)
        pixel_values = self.processor(image, return_tensors="pt").pixel_values
        # add labels (input_ids) by encoding the text
        labels = self.processor.tokenizer(text,
                                          padding="max_length",
                                          max_length=self.max_target_length).input_ids
        # important: make sure that PAD tokens are ignored by the loss function
        labels = [label if label != self.processor.tokenizer.pad_token_id else -100 for label in labels]

        encoding = {"file": file_name, "pixel_values": pixel_values.squeeze(), "labels": torch.tensor(labels)}
        return encoding
```

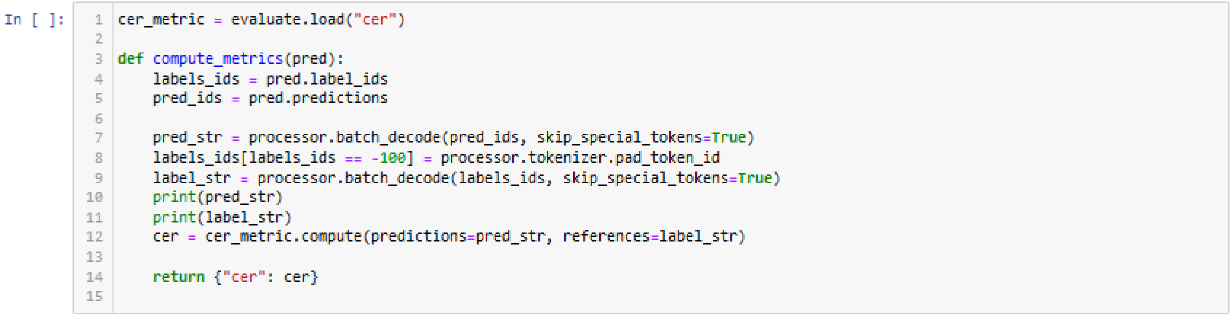
Figure A.2 Dataset class



```python
def get_model(model='microsoft/trocr-base-handwritten'):
    model = VisionEncoderDecoderModel.from_pretrained(model)
    # set special tokens used for creating the decoder_input_ids from the labels
    model.config.decoder_start_token_id = processor.tokenizer.cls_token_id
    model.config.pad_token_id = processor.tokenizer.pad_token_id
    # make sure vocab size is set correctly
    model.config.vocab_size = model.config.decoder.vocab_size

    # set to make it trainable:
    model.config.decoder.is_decoder = True
    model.config.decoder.add_cross_attention = True

    # set beam search parameters
    model.config.eos_token_id = processor.tokenizer.sep_token_id
    model.config.max_length = 64
    model.config.early_stopping = True
    model.config.no_repeat_ngram_size = 3
    model.config.length_penalty = 2.0
    model.config.num_beams = 4
    return model
```

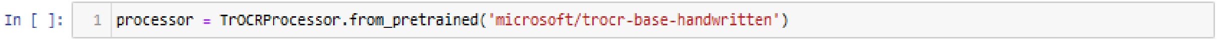
```python
root="/content/drive/MyDrive/Theses/Experiments/Historical/"
```

Figure A.3 Building the model



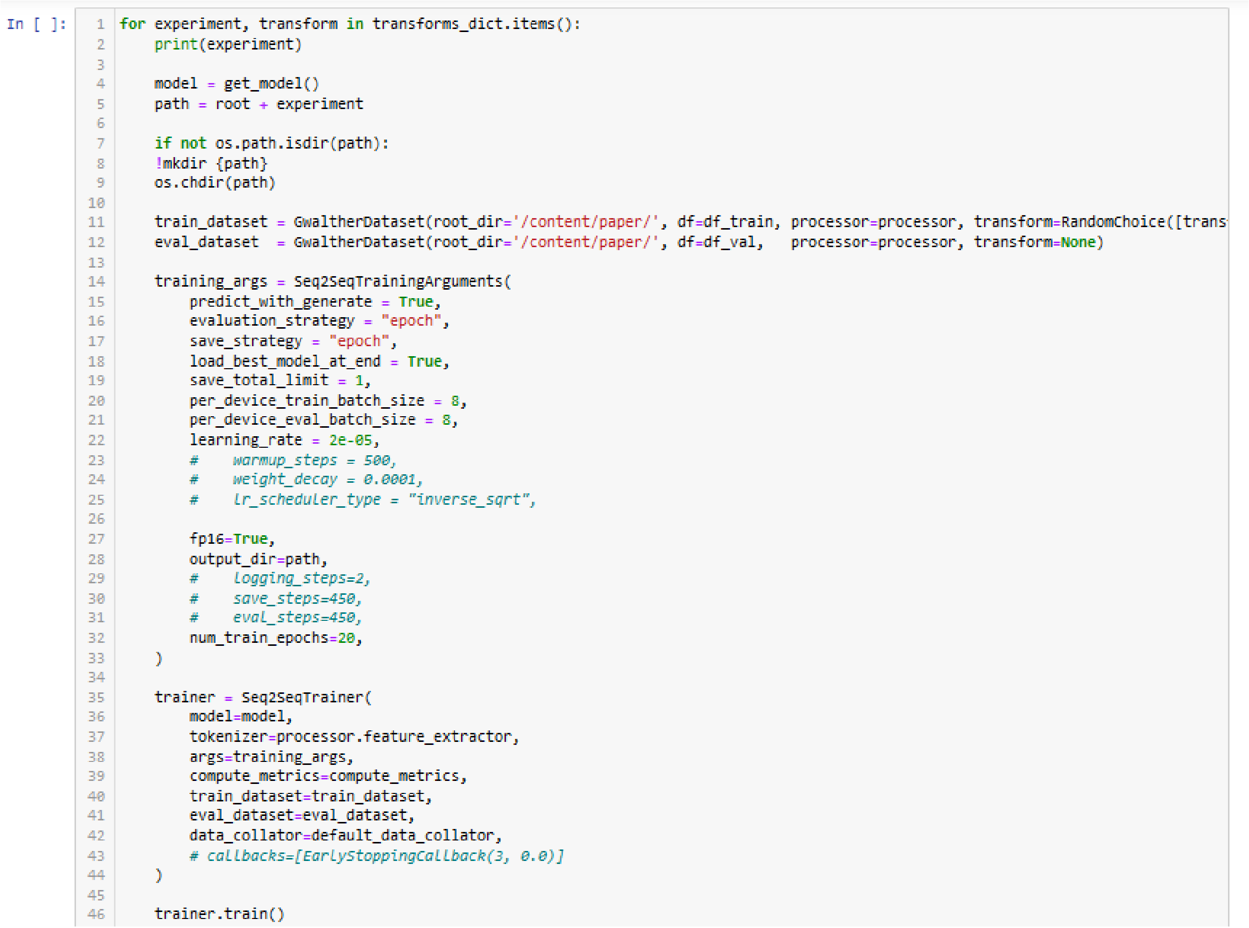
```
In [ ]:
cer_metric = evaluate.load("cer")

def compute_metrics(pred):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions

    pred_str = processor.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids == -100] = processor.tokenizer.pad_token_id
    label_str = processor.batch_decode(labels_ids, skip_special_tokens=True)
    print(pred_str)
    print(label_str)
    cer = cer_metric.compute(predictions=pred_str, references=label_str)

    return {"cer": cer}

```

Figure A.4 CER metric definition



```
In [ ]:
processor = TrOCRProcessor.from_pretrained('microsoft/trocr-base-handwritten')
```

Figure A.5 Creating the processor. Huggingface’s TrOCR tokenizer.



In [ ]:

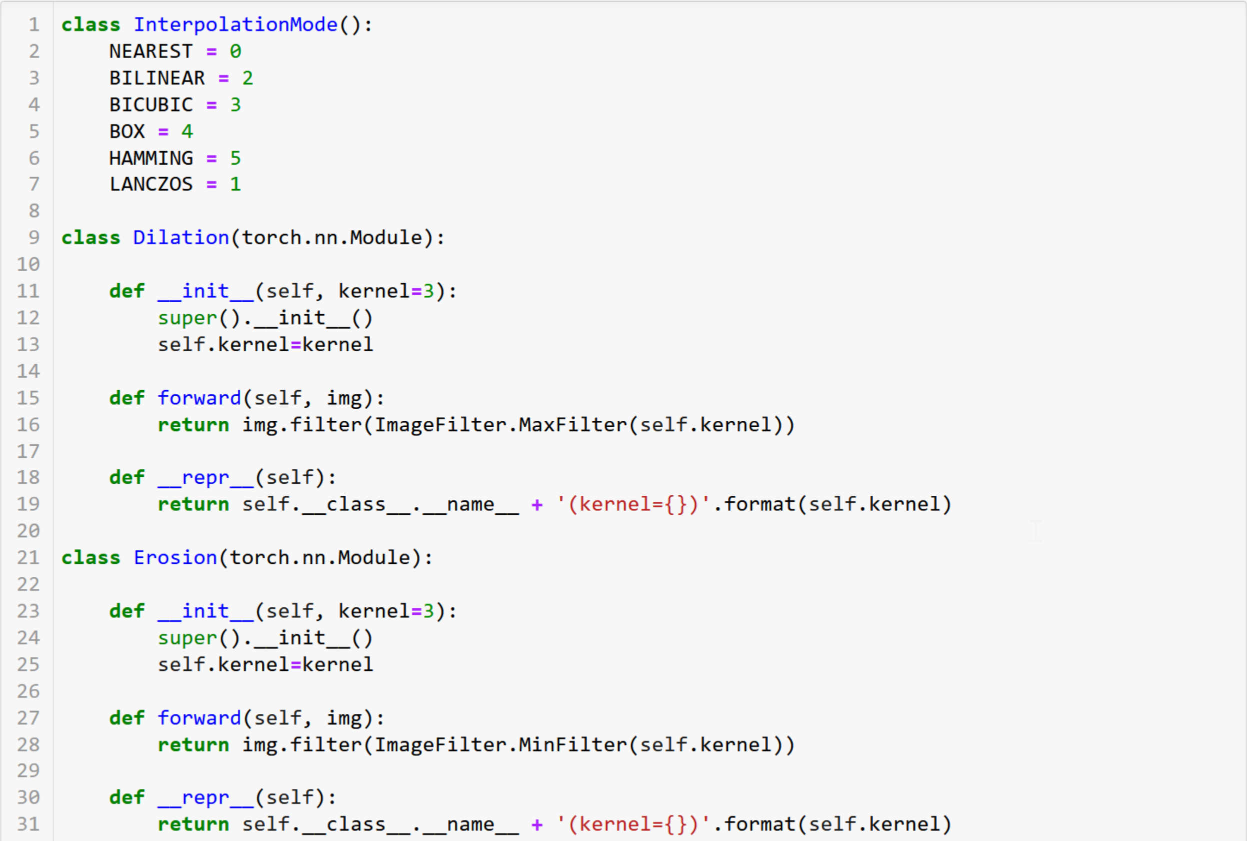
```
for experiment, transform in transforms_dict.items():
    print(experiment)

    model = get_model()
    path = root + experiment

    if not os.path.isdir(path):
    !mkdir {path}
    os.chdir(path)

    train_dataset = GwaltherDataset(root_dir='/content/paper/', df=df_train, processor=processor, transform=RandomChoice([trans
    eval_dataset  = GwaltherDataset(root_dir='/content/paper/', df=df_val,   processor=processor, transform=None)

    training_args = Seq2SeqTrainingArguments(
        predict_with_generate = True,
        evaluation_strategy = "epoch",
        save_strategy = "epoch",
        load_best_model_at_end = True,
        save_total_limit = 1,
        per_device_train_batch_size = 8,
        per_device_eval_batch_size = 8,
        learning_rate = 2e-05,
        #    warmup_steps = 500,
        #    weight_decay = 0.0001,
        #    lr_scheduler_type = "inverse_sqrt",

        fp16=True,
        output_dir=path,
        #    logging_steps=2,
        #    save_steps=450,
        #    eval_steps=450,
        num_train_epochs=20,
    )

    trainer = Seq2SeqTrainer(
        model=model,
        tokenizer=processor.feature_extractor,
        args=training_args,
        compute_metrics=compute_metrics,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        data_collator=default_data_collator,
        # callbacks=[EarlyStoppingCallback(3, 0.0)]
    )

    trainer.train()
```

Figure A.6 Training

APPENDIX B

AUGMENTATION CODE

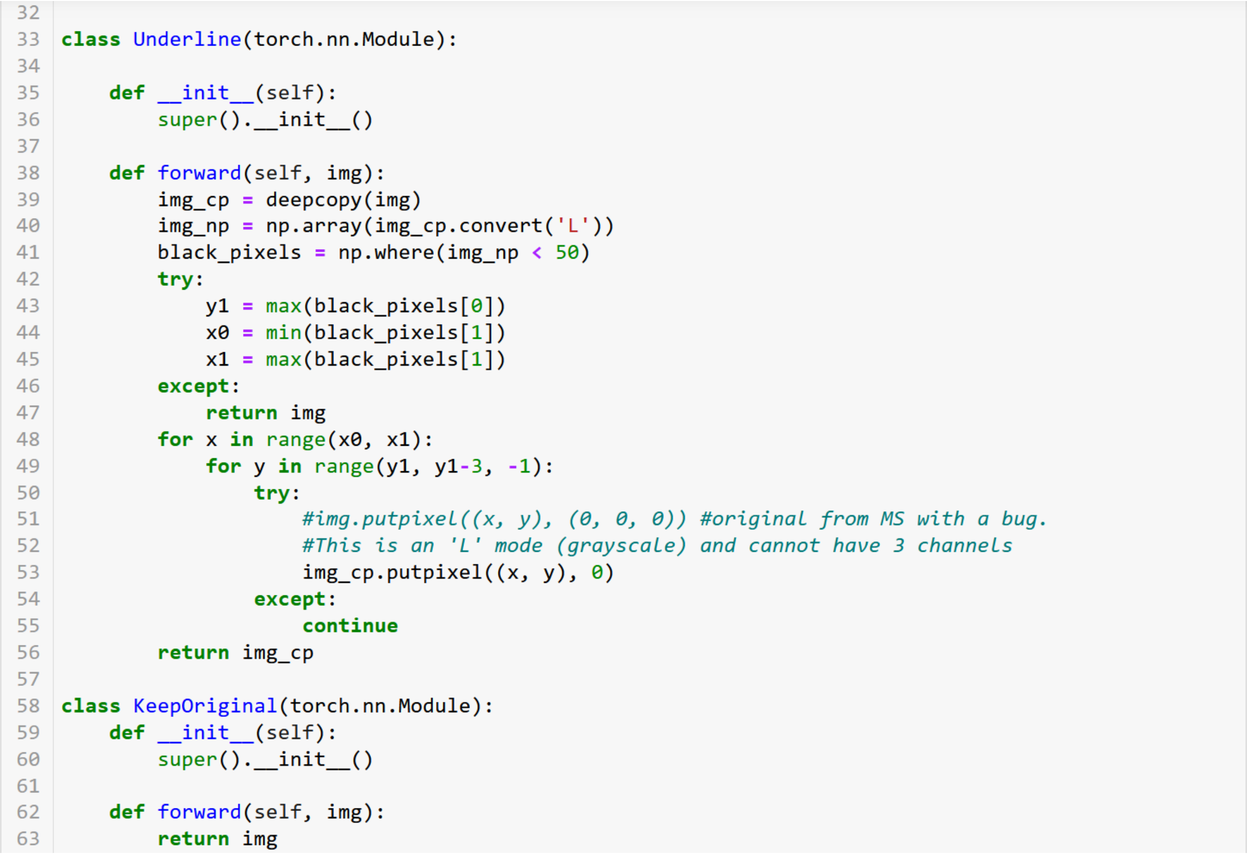
We used the following code to augment the images. Please note that to replicate the augmentation used in TrOCR's pre-training, we copied the code from the TrOCR source code[10] and fixed a few minor bugs.



```python
class InterpolationMode():
    NEAREST = 0
    BILINEAR = 2
    BICUBIC = 3
    BOX = 4
    HAMMING = 5
    LANCZOS = 1

class Dilation(torch.nn.Module):

    def __init__(self, kernel=3):
        super().__init__()
        self.kernel=kernel

    def forward(self, img):
        return img.filter(ImageFilter.MaxFilter(self.kernel))

    def __repr__(self):
        return self.__class__.__name__ + '(kernel={})'.format(self.kernel)

class Erosion(torch.nn.Module):

    def __init__(self, kernel=3):
        super().__init__()
        self.kernel=kernel

    def forward(self, img):
        return img.filter(ImageFilter.MinFilter(self.kernel))

    def __repr__(self):
        return self.__class__.__name__ + '(kernel={})'.format(self.kernel)
```

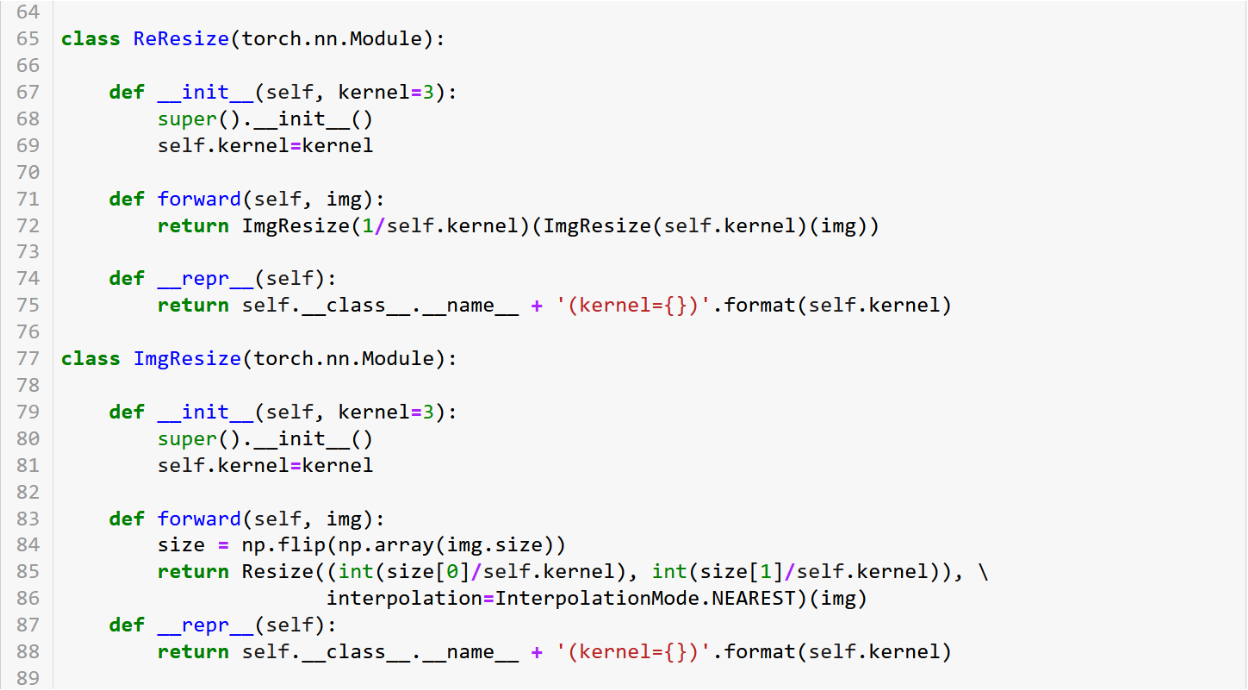
Figure B.1 TrOCR Dilation and Erosion code

[10] https://github.com/microsoft/unilm/tree/master/trocr



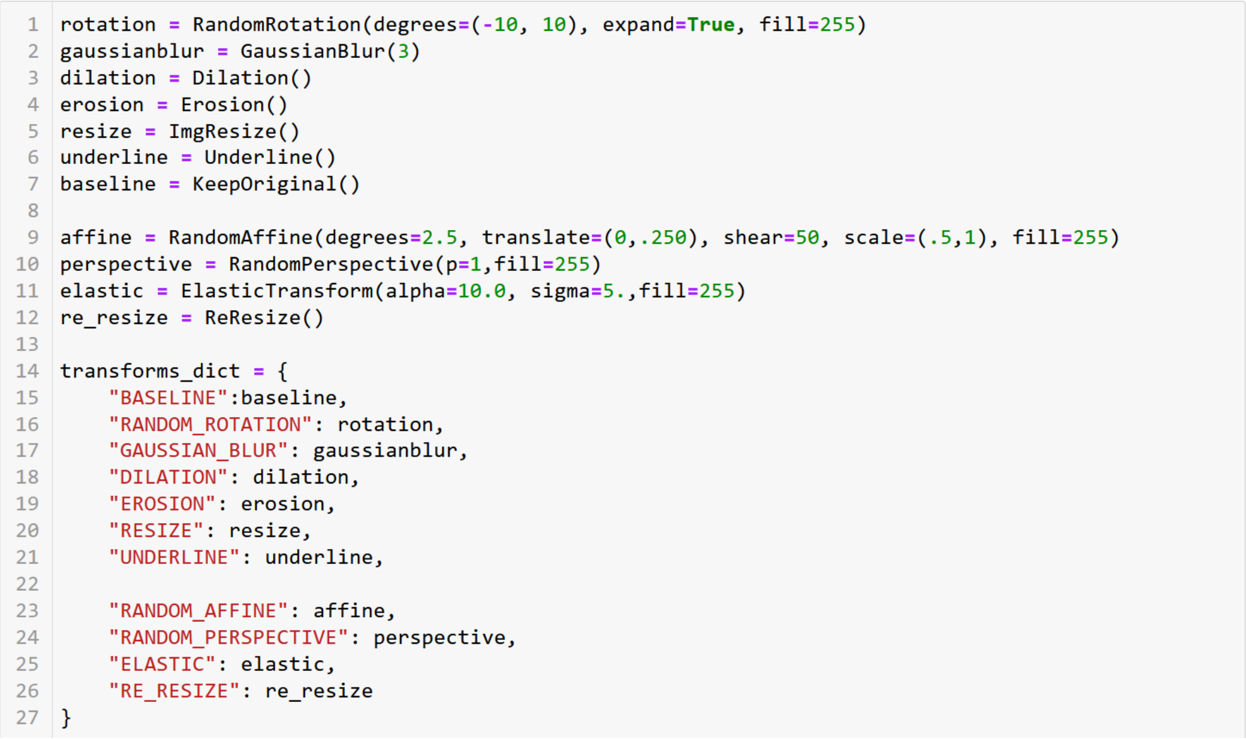
```python
class Underline(torch.nn.Module):

    def __init__(self):
        super().__init__()

    def forward(self, img):
        img_cp = deepcopy(img)
        img_np = np.array(img_cp.convert('L'))
        black_pixels = np.where(img_np < 50)
        try:
            y1 = max(black_pixels[0])
            x0 = min(black_pixels[1])
            x1 = max(black_pixels[1])
        except:
            return img
        for x in range(x0, x1):
            for y in range(y1, y1-3, -1):
                try:
                    #img.putpixel((x, y), (0, 0, 0)) #original from MS with a bug.
                    #This is an 'L' mode (grayscale) and cannot have 3 channels
                    img_cp.putpixel((x, y), 0)
                except:
                    continue
        return img_cp

class KeepOriginal(torch.nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, img):
        return img
```

Figure B.2 TrOCR’s Underline and KeepOriginal code



```
class ReResize(torch.nn.Module):

    def __init__(self, kernel=3):
        super().__init__()
        self.kernel=kernel

    def forward(self, img):
        return ImgResize(1/self.kernel)(ImgResize(self.kernel)(img))

    def __repr__(self):
        return self.__class__.__name__ + '(kernel={})'.format(self.kernel)

class ImgResize(torch.nn.Module):

    def __init__(self, kernel=3):
        super().__init__()
        self.kernel=kernel

    def forward(self, img):
        size = np.flip(np.array(img.size))
        return Resize((int(size[0]/self.kernel), int(size[1]/self.kernel)), \
                      interpolation=InterpolationMode.NEAREST)(img)
    def __repr__(self):
        return self.__class__.__name__ + '(kernel={})'.format(self.kernel)
```

Figure B.3 Image Resize and Re-Resize code



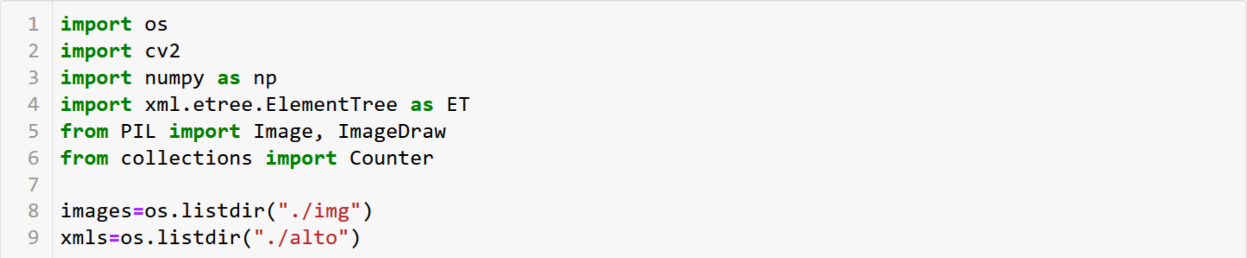
```python
rotation = RandomRotation(degrees=(-10, 10), expand=True, fill=255)
gaussianblur = GaussianBlur(3)
dilation = Dilation()
erosion = Erosion()
resize = ImgResize()
underline = Underline()
baseline = KeepOriginal()

affine = RandomAffine(degrees=2.5, translate=(0,.250), shear=50, scale=(.5,1), fill=255)
perspective = RandomPerspective(p=1,fill=255)
elastic = ElasticTransform(alpha=10.0, sigma=5.,fill=255)
re_resize = ReResize()

transforms_dict = {
    "BASELINE":baseline,
    "RANDOM_ROTATION": rotation,
    "GAUSSIAN_BLUR": gaussianblur,
    "DILATION": dilation,
    "EROSION": erosion,
    "RESIZE": resize,
    "UNDERLINE": underline,

    "RANDOM_AFFINE": affine,
    "RANDOM_PERSPECTIVE": perspective,
    "ELASTIC": elastic,
    "RE_RESIZE": re_resize
}
```

Figure B.4 More augmentations and creating the augmentation container (Python dictionary)

APPENDIX C

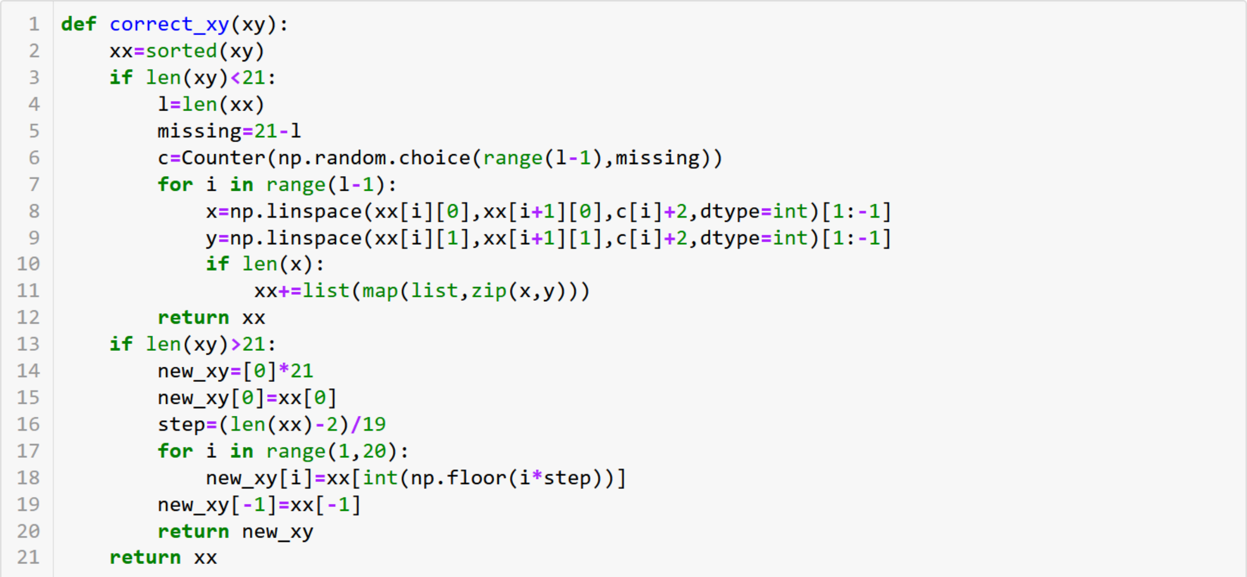
SOURCE CODE FOR PREPROCESSING AND CREATING THE LINE IMAGES DATASET

We used the following code to preprocess the images.



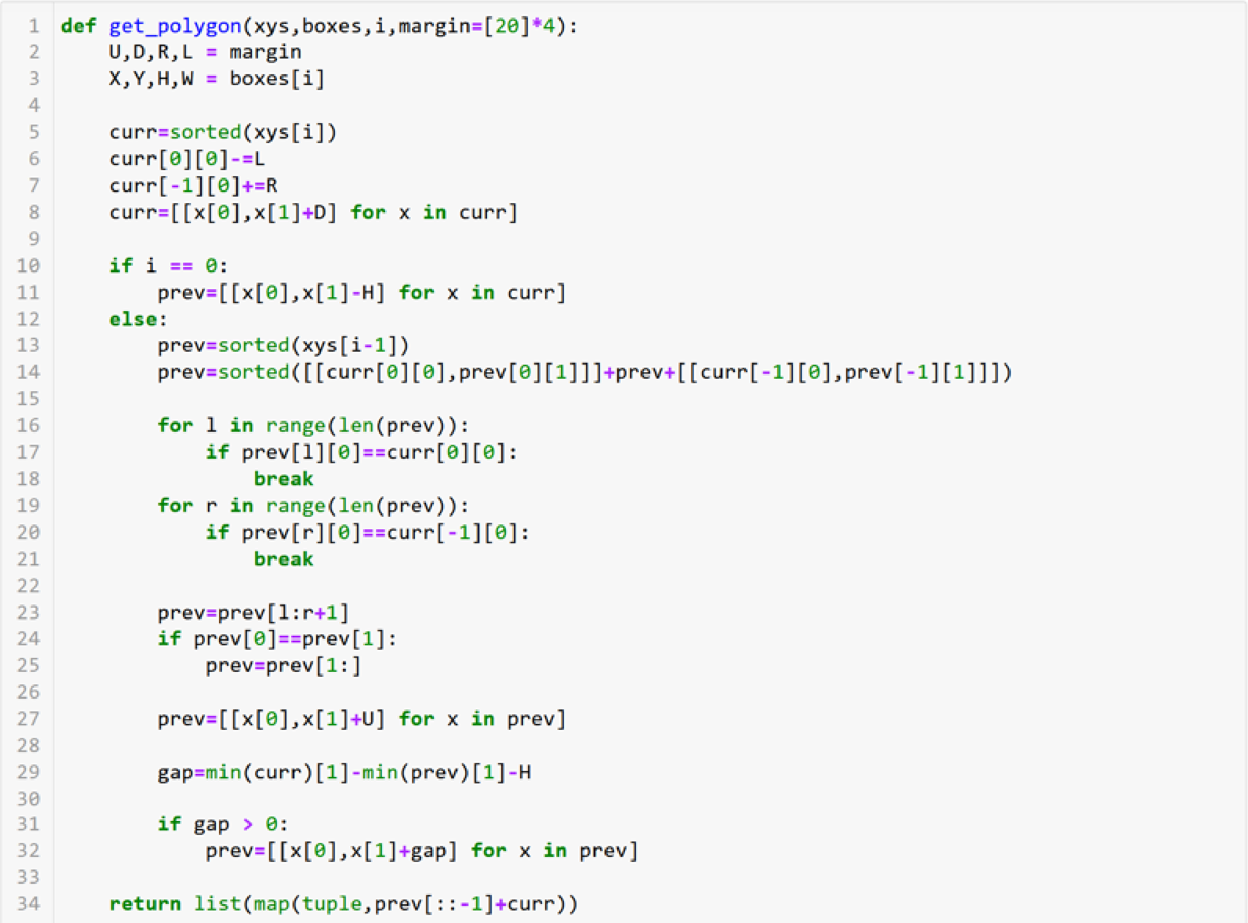
```python
import os
import cv2
import numpy as np
import xml.etree.ElementTree as ET
from PIL import Image, ImageDraw
from collections import Counter

images=os.listdir("./img")
xmls=os.listdir("./alto")
```

Figure C.1 Import section, followed by “global and useful” variables.



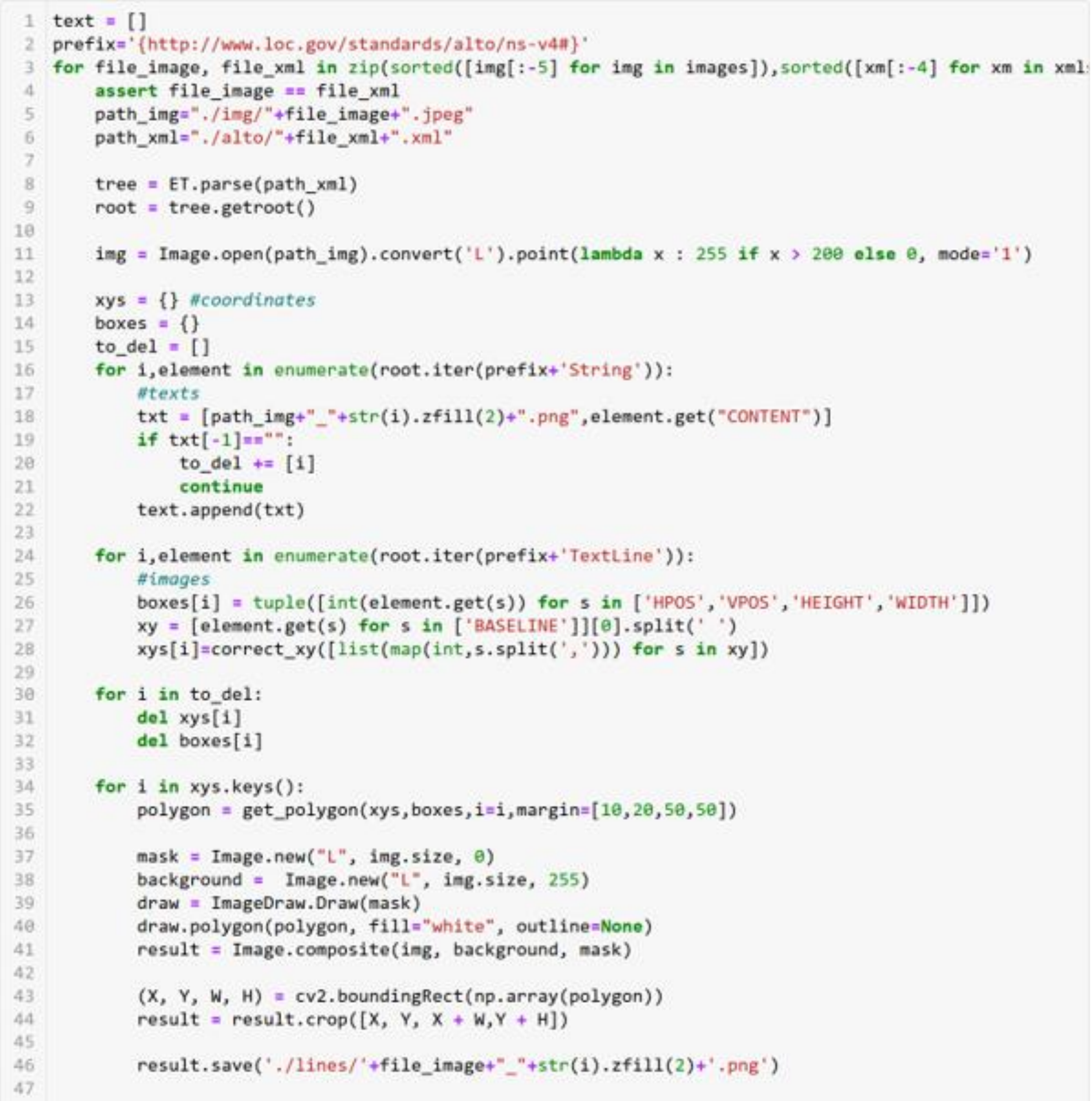
```python
def correct_xy(xy):
    xx=sorted(xy)
    if len(xy)<21:
        l=len(xx)
        missing=21-l
        c=Counter(np.random.choice(range(l-1),missing))
        for i in range(l-1):
            x=np.linspace(xx[i][0],xx[i+1][0],c[i]+2,dtype=int)[1:-1]
            y=np.linspace(xx[i][1],xx[i+1][1],c[i]+2,dtype=int)[1:-1]
            if len(x):
                xx+=list(map(list,zip(x,y)))
        return xx
    if len(xy)>21:
        new_xy=[0]*21
        new_xy[0]=xx[0]
        step=(len(xx)-2)/19
        for i in range(1,20):
            new_xy[i]=xx[int(np.floor(i*step))]
        new_xy[-1]=xx[-1]
        return new_xy
    return xx
```

Figure C.2 Function to split the original image into strict line images



```python
def get_polygon(xys,boxes,i,margin=[20]*4):
    U,D,R,L = margin
    X,Y,H,W = boxes[i]

    curr=sorted(xys[i])
    curr[0][0]-=L
    curr[-1][0]+=R
    curr=[[x[0],x[1]+D] for x in curr]

    if i == 0:
        prev=[[x[0],x[1]-H] for x in curr]
    else:
        prev=sorted(xys[i-1])
        prev=sorted([[curr[0][0],prev[0][1]]]+prev+[[curr[-1][0],prev[-1][1]]])

        for l in range(len(prev)):
            if prev[l][0]==curr[0][0]:
                break
        for r in range(len(prev)):
            if prev[r][0]==curr[-1][0]:
                break

        prev=prev[l:r+1]
        if prev[0]==prev[1]:
            prev=prev[1:]

        prev=[[x[0],x[1]+U] for x in prev]

        gap=min(curr)[1]-min(prev)[1]-H

        if gap > 0:
            prev=[[x[0],x[1]+gap] for x in prev]

    return list(map(tuple,prev[::-1]+curr))
```

Figure C.3 Getting the correct coordinates around every line



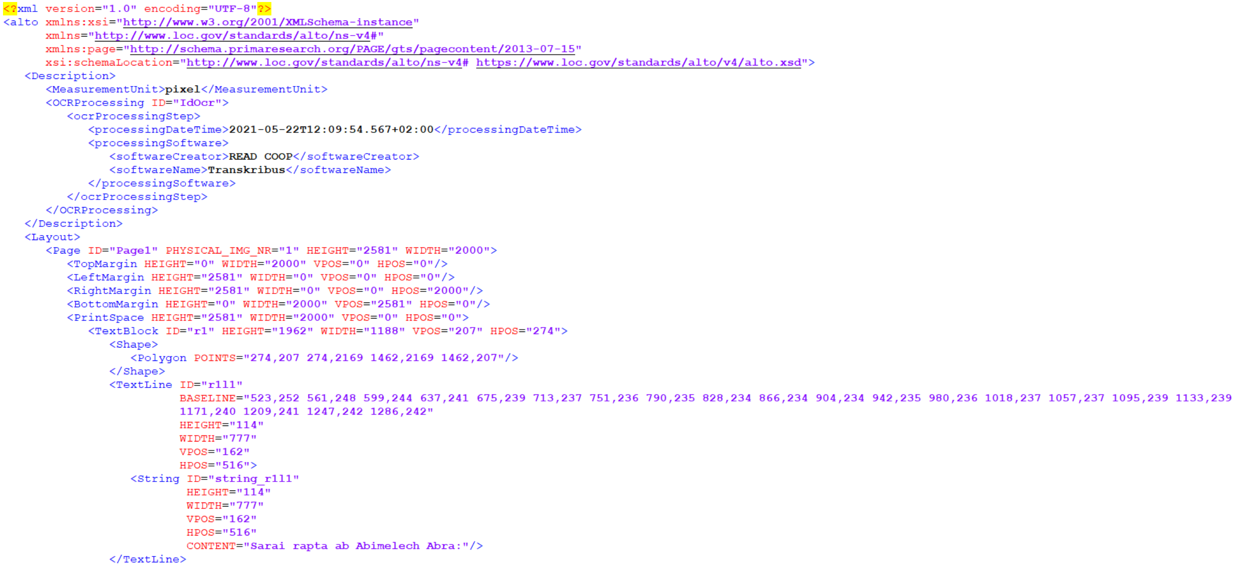
```
text = []
prefix='{http://www.loc.gov/standards/alto/ns-v4#}'
for file_image, file_xml in zip(sorted([img[:-5] for img in images]),sorted([xm[:-4] for xm in xml
    assert file_image == file_xml
    path_img="./img/"+file_image+".jpeg"
    path_xml="./alto/"+file_xml+".xml"

    tree = ET.parse(path_xml)
    root = tree.getroot()

    img = Image.open(path_img).convert('L').point(lambda x : 255 if x > 200 else 0, mode='1')

    xys = {} #coordinates
    boxes = {}
    to_del = []
    for i,element in enumerate(root.iter(prefix+'String')):
        #texts
        txt = [path_img+"_"+str(i).zfill(2)+".png",element.get("CONTENT")]
        if txt[-1]=="":
            to_del += [i]
            continue
        text.append(txt)

    for i,element in enumerate(root.iter(prefix+'TextLine')):
        #images
        boxes[i] = tuple([int(element.get(s)) for s in ['HPOS','VPOS','HEIGHT','WIDTH']])
        xy = [element.get(s) for s in ['BASELINE']][0].split(' ')
        xys[i]=correct_xy([list(map(int,s.split(','))) for s in xy])

    for i in to_del:
        del xys[i]
        del boxes[i]

    for i in xys.keys():
        polygon = get_polygon(xys,boxes,i=i,margin=[10,20,50,50])

        mask = Image.new("L", img.size, 0)
        background =  Image.new("L", img.size, 255)
        draw = ImageDraw.Draw(mask)
        draw.polygon(polygon, fill="white", outline=None)
        result = Image.composite(img, background, mask)

        (X, Y, W, H) = cv2.boundingRect(np.array(polygon))
        result = result.crop([X, Y, X + W,Y + H])

        result.save('./lines/'+file_image+"_"+str(i).zfill(2)+'.png')

```

Figure C.4 Main code to prepare the line images

APPENDIX D

METADATA XML FILE USED TO CREATE THE LINE IMAGES

XML file used to get the coordinates of lines.



```
<?xml version="1.0" encoding="UTF-8"?>
<alto xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.loc.gov/standards/alto/ns-v4#"
      xmlns:page="http://schema.primaresearch.org/PAGE/gts/pagecontent/2013-07-15"
      xsi:schemaLocation="http://www.loc.gov/standards/alto/ns-v4# https://www.loc.gov/standards/alto/v4/alto.xsd">
   <Description>
      <MeasurementUnit>pixel</MeasurementUnit>
      <OCRProcessing ID="IdOcr">
         <ocrProcessingStep>
            <processingDateTime>2021-05-22T12:09:54.567+02:00</processingDateTime>
            <processingSoftware>
               <softwareCreator>READ COOP</softwareCreator>
               <softwareName>Transkribus</softwareName>
            </processingSoftware>
         </ocrProcessingStep>
      </OCRProcessing>
   </Description>
   <Layout>
      <Page ID="Page1" PHYSICAL_IMG_NR="1" HEIGHT="2581" WIDTH="2000">
         <TopMargin HEIGHT="0" WIDTH="2000" VPOS="0" HPOS="0"/>
         <LeftMargin HEIGHT="2581" WIDTH="0" VPOS="0" HPOS="0"/>
         <RightMargin HEIGHT="2581" WIDTH="0" VPOS="0" HPOS="2000"/>
         <BottomMargin HEIGHT="0" WIDTH="2000" VPOS="2581" HPOS="0"/>
         <PrintSpace HEIGHT="2581" WIDTH="2000" VPOS="0" HPOS="0">
            <TextBlock ID="r1" HEIGHT="1962" WIDTH="1188" VPOS="207" HPOS="274">
               <Shape>
                  <Polygon POINTS="274,207 274,2169 1462,2169 1462,207"/>
               </Shape>
               <TextLine ID="r1l1"
                         BASELINE="523,252 561,248 599,244 637,241 675,239 713,237 751,236 790,235 828,234 866,234 904,234 942,235 980,236 1018,237 1057,237 1095,239 1133,239
                         1171,240 1209,241 1247,242 1286,242"
                         HEIGHT="114"
                         WIDTH="777"
                         VPOS="162"
                         HPOS="516">
                  <String ID="string_r1l1"
                          HEIGHT="114"
                          WIDTH="777"
                          VPOS="162"
                          HPOS="516"
                          CONTENT="Sarai rapta ab Abimelech Abra:"/>
               </TextLine>
```

Figure D.1 The first row of file 1111637

APPENDIX E

COMPLETE TRAINING AND VALIDATION LOSS

Table E.1 The complete output of the training and validation loss

Row	epoch	Training Loss	Validation Loss	CER	Augmentation
0	1	1.078	0.519562	0.06008	Baseline
1	2	1.078	0.403055	0.041943	Baseline
2	3	0.4087	0.364726	0.037409	Baseline
3	4	0.2467	0.343347	0.032704	Baseline
4	5	0.1777	0.313211	0.03055	Baseline
5	6	0.1181	0.316009	0.029984	Baseline
6	7	0.0881	0.338562	0.02987	Baseline
7	8	0.067	0.343294	0.026299	Baseline
8	9	0.0527	0.319898	0.025279	Baseline
9	10	0.0325	0.311561	0.024939	Baseline
10	11	0.0325	0.318875	0.024032	Baseline
11	12	0.0197	0.33979	0.024089	Baseline
12	13	0.015	0.300684	0.021482	Baseline
13	14	0.008	0.305111	0.021878	Baseline
14	15	0.007	0.302782	0.021822	Baseline
15	16	0.0033	0.307577	0.021425	Baseline
16	17	0.0023	0.30581	0.021595	Baseline
17	18	0.0017	0.304262	0.019781	Baseline
18	19	0.0006	0.305313	0.019101	Baseline
19	20	0.0004	0.306508	0.019271	Baseline
20	1	1.1432	0.527584	0.05651	Random Rotation
21	2	1.1432	0.383074	0.040186	Random Rotation
22	3	0.4447	0.346119	0.035595	Random Rotation
23	4	0.2715	0.326685	0.032307	Random Rotation
24	5	0.1989	0.350332	0.032761	Random Rotation
25	6	0.1571	0.313284	0.028907	Random Rotation
26	7	0.1129	0.303798	0.027716	Random Rotation
27	8	0.087	0.316333	0.027943	Random Rotation
28	9	0.0643	0.314933	0.025449	Random Rotation
29	10	0.0449	0.291711	0.024202	Random Rotation
30	11	0.0449	0.290608	0.023239	Random Rotation
31	12	0.0297	0.284602	0.022615	Random Rotation
32	13	0.0207	0.297219	0.023919	Random Rotation
33	14	0.0154	0.28809	0.023239	Random Rotation
34	15	0.0081	0.282304	0.020008	Random Rotation
35	16	0.0062	0.290071	0.020858	Random Rotation
36	17	0.0049	0.29592	0.022048	Random Rotation
37	18	0.003	0.296295	0.019838	Random Rotation
38	19	0.0021	0.285791	0.018818	Random Rotation
39	20	0.0012	0.283552	0.018591	Random Rotation

Table E.1 (continued)

Row	**epoch**	**Training Loss**	**Validation Loss**	**CER**	**Augmentation**
40	1	1.0643	0.477105	0.052145	Gaussian Blur
41	2	1.0643	0.391859	0.041206	Gaussian Blur
42	3	0.3901	0.350529	0.036218	Gaussian Blur
43	4	0.2508	0.32103	0.029133	Gaussian Blur
44	5	0.173	0.344791	0.031911	Gaussian Blur
45	6	0.1206	0.33	0.030664	Gaussian Blur
46	7	0.0832	0.335416	0.028056	Gaussian Blur
47	8	0.0617	0.325686	0.026413	Gaussian Blur
48	9	0.049	0.307256	0.02902	Gaussian Blur
49	10	0.0315	0.307448	0.024429	Gaussian Blur
50	11	0.0315	0.305244	0.026639	Gaussian Blur
51	12	0.0224	0.326817	0.026469	Gaussian Blur
52	13	0.012	0.302345	0.022955	Gaussian Blur
53	14	0.0072	0.314744	0.022048	Gaussian Blur
54	15	0.0039	0.300724	0.022218	Gaussian Blur
55	16	0.0036	0.299853	0.021085	Gaussian Blur
56	17	0.0012	0.296029	0.020065	Gaussian Blur
57	18	0.0027	0.311083	0.020348	Gaussian Blur
58	19	0.0014	0.311263	0.019951	Gaussian Blur
59	20	0.0004	0.312543	0.020405	Gaussian Blur
60	1	1.1166	0.565416	0.068469	Dilation
61	2	1.1166	0.476236	0.051182	Dilation
62	3	0.4186	0.399126	0.045797	Dilation
63	4	0.2623	0.43333	0.047611	Dilation
64	5	0.1794	0.376297	0.039109	Dilation
65	6	0.1214	0.366847	0.037125	Dilation
66	7	0.091	0.360929	0.034405	Dilation
67	8	0.0645	0.395488	0.034235	Dilation
68	9	0.0468	0.352025	0.031967	Dilation
69	10	0.0322	0.345662	0.031457	Dilation
70	11	0.0322	0.345543	0.031004	Dilation
71	12	0.0198	0.362451	0.031231	Dilation
72	13	0.0138	0.348637	0.029247	Dilation
73	14	0.0059	0.349265	0.026129	Dilation
74	15	0.005	0.342616	0.028	Dilation
75	16	0.0031	0.345011	0.028113	Dilation
76	17	0.0016	0.336828	0.022445	Dilation
77	18	0.0013	0.337236	0.022955	Dilation
78	19	0.0009	0.334826	0.023182	Dilation
79	20	0.0007	0.339993	0.023125	Dilation
80	1	1.1638	0.613491	0.07374	Erosion

Table E.1 (continued)

Row	**epoch**	**Training Loss**	**Validation Loss**	**CER**	**Augmentation**
81	2	1.1638	0.435962	0.047101	Erosion
82	3	0.4306	0.471658	0.052996	Erosion
83	4	0.2707	0.392846	0.035992	Erosion
84	5	0.1916	0.346806	0.034858	Erosion
85	6	0.1307	0.353428	0.034461	Erosion
86	7	0.094	0.357213	0.033611	Erosion
87	8	0.0693	0.369908	0.034858	Erosion
88	9	0.0487	0.353795	0.02936	Erosion
89	10	0.0314	0.350402	0.02868	Erosion
90	11	0.0314	0.343966	0.028056	Erosion
91	12	0.0197	0.352663	0.02715	Erosion
92	13	0.0125	0.357017	0.02749	Erosion
93	14	0.0069	0.365086	0.026639	Erosion
94	15	0.0054	0.372907	0.025733	Erosion
95	16	0.0031	0.367379	0.025959	Erosion
96	17	0.0015	0.355053	0.023862	Erosion
97	18	0.0011	0.355267	0.024202	Erosion
98	19	0.0008	0.364577	0.024202	Erosion
99	20	0.0004	0.361567	0.023295	Erosion
100	1	1.3185	0.55012	0.061101	Resize
101	2	1.3185	0.42272	0.04353	Resize
102	3	0.4627	0.36906	0.038656	Resize
103	4	0.2895	0.36019	0.037522	Resize
104	5	0.202	0.322729	0.035935	Resize
105	6	0.1315	0.313287	0.032081	Resize
106	7	0.0945	0.334587	0.029473	Resize
107	8	0.0678	0.379378	0.030777	Resize
108	9	0.0525	0.353082	0.030777	Resize
109	10	0.0347	0.330066	0.02749	Resize
110	11	0.0347	0.332629	0.026583	Resize
111	12	0.0216	0.334915	0.027263	Resize
112	13	0.0139	0.331957	0.025506	Resize
113	14	0.0096	0.322734	0.025449	Resize
114	15	0.0072	0.320495	0.024146	Resize
115	16	0.0044	0.326471	0.024939	Resize
116	17	0.0026	0.325962	0.024032	Resize
117	18	0.0024	0.327345	0.023125	Resize
118	19	0.0008	0.325227	0.022559	Resize
119	20	0.0009	0.327641	0.023069	Resize
120	1	1.1076	0.527811	0.057643	Underline
121	2	1.1076	0.399528	0.042793	Underline

Table E.1 (continued)

Row	epoch	Training Loss	Validation Loss	CER	Augmentation
122	3	0.4062	0.389996	0.04285	Underline
123	4	0.2513	0.356835	0.034291	Underline
124	5	0.1755	0.311959	0.031741	Underline
125	6	0.1176	0.319518	0.029417	Underline
126	7	0.0923	0.319267	0.028226	Underline
127	8	0.0641	0.328906	0.026129	Underline
128	9	0.0459	0.324883	0.026526	Underline
129	10	0.0299	0.318598	0.025052	Underline
130	11	0.0299	0.337376	0.025506	Underline
131	12	0.0194	0.313705	0.023749	Underline
132	13	0.014	0.316792	0.023976	Underline
133	14	0.0078	0.318613	0.021935	Underline
134	15	0.0052	0.320133	0.022672	Underline
135	16	0.0043	0.320846	0.022332	Underline
136	17	0.0029	0.317413	0.022388	Underline
137	18	0.0025	0.313905	0.020575	Underline
138	19	0.0014	0.312034	0.020801	Underline
139	20	0.0007	0.312286	0.020348	Underline
140	1	1.3664	0.522218	0.057303	Random Affine
141	2	1.3664	0.39962	0.043247	Random Affine
142	3	0.608	0.388454	0.042907	Random Affine
143	4	0.427	0.327208	0.034178	Random Affine
144	5	0.3178	0.318385	0.032818	Random Affine
145	6	0.2555	0.415137	0.053222	Random Affine
146	7	0.1959	0.294005	0.029984	Random Affine
147	8	0.1531	0.28361	0.028567	Random Affine
148	9	0.1239	0.316811	0.031174	Random Affine
149	10	0.099	0.273864	0.025563	Random Affine
150	11	0.099	0.276069	0.027546	Random Affine
151	12	0.0756	0.283561	0.025846	Random Affine
152	13	0.0594	0.273119	0.024599	Random Affine
153	14	0.0476	0.290156	0.024939	Random Affine
154	15	0.0417	0.274205	0.024486	Random Affine
155	16	0.0297	0.266478	0.021028	Random Affine
156	17	0.027	0.279343	0.022332	Random Affine
157	18	0.0207	0.27608	0.022445	Random Affine
158	19	0.015	0.274905	0.021595	Random Affine
159	20	0.0141	0.274926	0.021312	Random Affine
160	1	1.6312	0.581474	0.068299	Random Perspective
161	2	1.6312	0.43874	0.048915	Random Perspective
162	3	0.7344	0.414392	0.046817	Random Perspective

Table E.1 (continued)

Row	epoch	Training Loss	Validation Loss	CER	Augmentation
163	4	0.4938	0.350126	0.036898	Random Perspective
164	5	0.3468	0.349374	0.038826	Random Perspective
165	6	0.2842	0.315118	0.032818	Random Perspective
166	7	0.2213	0.326544	0.032874	Random Perspective
167	8	0.1678	0.336331	0.031287	Random Perspective
168	9	0.1371	0.303089	0.028283	Random Perspective
169	10	0.1058	0.289244	0.027886	Random Perspective
170	11	0.1058	0.304415	0.028793	Random Perspective
171	12	0.0815	0.311864	0.027206	Random Perspective
172	13	0.0633	0.307015	0.026186	Random Perspective
173	14	0.0504	0.301629	0.025789	Random Perspective
174	15	0.039	0.296656	0.023409	Random Perspective
175	16	0.0292	0.29267	0.024712	Random Perspective
176	17	0.0249	0.287982	0.023012	Random Perspective
177	18	0.0172	0.285475	0.022729	Random Perspective
178	19	0.0139	0.28892	0.022899	Random Perspective
179	20	0.0098	0.287999	0.022729	Random Perspective
180	1	1.2867	0.515767	0.05651	Elastic
181	2	1.2867	0.406968	0.042566	Elastic
182	3	0.4896	0.385982	0.037522	Elastic
183	4	0.3174	0.340185	0.031344	Elastic
184	5	0.2153	0.324556	0.031797	Elastic
185	6	0.1592	0.339001	0.031797	Elastic
186	7	0.1181	0.333239	0.032024	Elastic
187	8	0.0877	0.328647	0.028	Elastic
188	9	0.0672	0.321553	0.025789	Elastic
189	10	0.0449	0.315571	0.025449	Elastic
190	11	0.0449	0.31439	0.026073	Elastic
191	12	0.035	0.326755	0.024202	Elastic
192	13	0.0246	0.330861	0.024146	Elastic
193	14	0.0202	0.310808	0.023295	Elastic
194	15	0.0122	0.301151	0.021822	Elastic
195	16	0.0092	0.292605	0.019044	Elastic
196	17	0.0074	0.3003	0.020121	Elastic
197	18	0.0038	0.302104	0.019044	Elastic
198	19	0.0024	0.299638	0.018591	Elastic
199	20	0.0018	0.297659	0.018591	Elastic
200	1	1.2119	0.491272	0.052542	Re Resize
201	2	1.2119	0.399982	0.04336	Re Resize
202	3	0.4672	0.353607	0.035085	Re Resize
203	4	0.2855	0.352745	0.032421	Re Resize

Table E.1 (continued)

Row	epoch	Training Loss	Validation Loss	CER	Augmentation
204	5	0.1872	0.321231	0.032194	Re Resize
205	6	0.1377	0.357808	0.031627	Re Resize
206	7	0.0962	0.330003	0.029417	Re Resize
207	8	0.0646	0.345332	0.02953	Re Resize
208	9	0.053	0.318062	0.026129	Re Resize
209	10	0.0287	0.324819	0.02834	Re Resize
210	11	0.0287	0.333619	0.025506	Re Resize
211	12	0.0243	0.323817	0.026186	Re Resize
212	13	0.0168	0.326259	0.023125	Re Resize
213	14	0.008	0.32286	0.021708	Re Resize
214	15	0.0073	0.331472	0.022729	Re Resize
215	16	0.0041	0.327241	0.020915	Re Resize
216	17	0.0019	0.324864	0.020575	Re Resize
217	18	0.001	0.330449	0.020971	Re Resize
218	19	0.0007	0.330838	0.020858	Re Resize
219	20	0.0004	0.329976	0.020858	Re Resize