university of groningen / faculty of science and engineering / artificial intelligence

Handwriting Recognition Project: Recognition of DDS and IAM

Thijs Lukkien, s3978389, t.lukkien.1@student.rug.nl,
Rob van der Zee, s2740133, r.j.van.der.zee.1@student.rug.nl
Otto Bervoets, s3417522, a.g.b.bervoets@student.rug.nl
Roben de Lange, s3799174 , r.de.lange.6@student.rug.nl
Group 1

Abstract: Optical Character Recognition (OCR) remains an active field of research in the AI community. In this project two handwriting recognition systems are designed for two different data sets using two different sets of techniques. Firstly, an OCR pipeline is designed for the Dead Sea Scrolls (DSS). With the lack of labeled data in this data set, the pipeline uses a set of carefully crafted character extraction methods combined with line segmentation. The extracted characters are then fed into an CNN recognizer that is trained on labeled character data. The second pipeline concerns the IAM dataset. Specifically a subset containing images of lines of text. This data is labeled and hence a Transformer based ORC model is used, derived from TrOCR with the addition of pre-training and data augmentations. The first pipeline has a Levenshtein distance that is between 36% - 40% less than an empty string. The best IAM pipeline has a word error rate of 17% and an character error rate of 10%.

1 Introduction and Literature review

For thousands of years, humans have written down texts. Optical Character Recognition (OCR) tries to translate written texts into machine-encoded text. The importance of this is threefold. Firstly it allows us to preserve and easily share written text. Secondly, it allows analysis of the text using computer algorithms. Lastly writing text is still a very natural thing for humans to do, and being able to properly translate this into machine-encoded text enhances our communication with computers.

One of the most ancient writings are those in the Dead Sea Scrolls (DSS) (Wise et al., 1996). These scrolls originate somewhere from 300-700 B.C. and contain both biblical and non-biblical texts. All scrolls are written in Hebrew and they have been found between 1946 and 1956. The scrolls have been intensively studied since their discovery, amongst others by the field of OCR. This report will also try to build a pipeline to recognise the text written in these scrolls.

A more recent labeled text dataset is the IAM dataset Marti & Bunke (1999). In this research, we will focus on a subset of the IAM dataset containing images of individual sentences. For this dataset, a second pipeline to recognize the text written in these images will be designed. So there will be two different pipelines, for two different datasets. What is also different between the two datasets is that the IAM dataset is labeled, whereas the DSS dataset not.

Let us now discuss some previous literature on this subject. We divided the previous work into roughly two categories. The first category is careful to feature/character extraction methods that use multiple more classical Machine Learning (ML) techniques to finally obtain the written text. The second category contains bigger end-to-end models, that do not rely on these carefully crafted algorithms.

To start with OCR systems that use carefully crafted character extraction methods we will discuss techniques described in Khorsheed (2002), a review paper concerning off-line Arabic character recognition. The authors describe various techniques that can be used to segment an image of

a written text into individual characters. We will highlight a couple of their techniques. First of all, we would like to highlight the technique of connected components. This technique inspects which pixels are connected, and a blob of connected pixels then is one connected component. In the Arabic written language, words consist of connected primitives. Hence, words are most of the time one connected component, this can however be different when there is a lot of noise in the image. Secondly, to segment these images a technique called vertical projection is described. In this technique, the pixel values will all be projected on the x-axis. Based on (global) minima that emerge in the graph that corresponds to this technique one can split the image into multiple letters. Thirdly, the paper also describes smoothing techniques to reduce noise in images. Both filtering techniques either fill small gaps between/within objects (Dilation) within images or open small gaps between two nearly touching objects (Erosion). The authors of Khorsheed (2002) do introduce neural networks to classify individual characters, however these were not as well developed as they are now, and more recent papers are more relevant.

On the topic of classifying individual characters we turn to the work of Tobing et al. (2022). The authors manually segmented and labelled the characters from the DSS. Then the authors used different convolutional neural networks on the segmented characters with performances ranging between 89-97 percent, where the best model was a ResNet (He et al., 2015) model pre-trained on ImageNet (Deng et al., 2009).

Let us now turn to an end-to-end OCR system. The authors of Li et al. (2023) propose such an end-to-end solution. The model proposed by the authors is a simple transformer based encoder-decoder model that achieves state of the art performance on handwritten text. The authors propose several models, in this paper we will focus on the base model of which the encoder layer is initialized with the weights from BERT Pre-Training of Image Transformers (BEit) Bao et al. (2022) and the decoder layer is initialized from A Robustly Optimized BERT Pretraining Approach (RoBERTA) Liu et al. (2019). This model needs to be fine tuned on the target dataset.

For training we will two techniques. First of all we will use pertaining, which will be discussed further in the methodology. Secondly Iceland & Kanan (2023) suggest that augmentations almost always benefit the performance of a model. Hence, we will also add this to the training step of our pipeline. We will discuss this further in Section 2.2.

2 Methodology - Dead Sea Scrolls

This pipeline will use mostly classical machine learning techniques instead of an end-to-end Deep Learning solution when processing Hebrew characters from the scrolls. The motivation behind this is mostly the small number of datapoints in this dataset. The input of this pipeline is a binarized image of the scroll, the output must be the text on the scroll. First, the data will be discussed in Section 2.1, followed by a number of sections dedicated to pipeline segments. The pipeline consists of the following segments:

1. Connected components analysis
2. Connected components analysis on blurred leftovers
3. Splitting connected components using vertical projection
4. Line segmentation
5. Classification per character

The first four items are part of our segmentation process, this is discussed in Section 2.3. The final step where individual characters are classified is discussed in Section 2.4.

Parts of the implementations of these methods were realized with the help of ChatGPT OpenAI (2023).

2.1 Data

There are several datasets available and used for this pipeline. First of all, there is an unlabeled dataset containing images of scrolls. There are in total 20 of these scrolls, of each scroll there is an original RGB-coloured JPG file, a grayscale version of this file, and lastly the binarized and cleaned version of this file. The binarized file is the version that will be used in this report. This binarized file has a background value of 1 and a signal value of 0. As advised in the lecture, we invert the image to feed

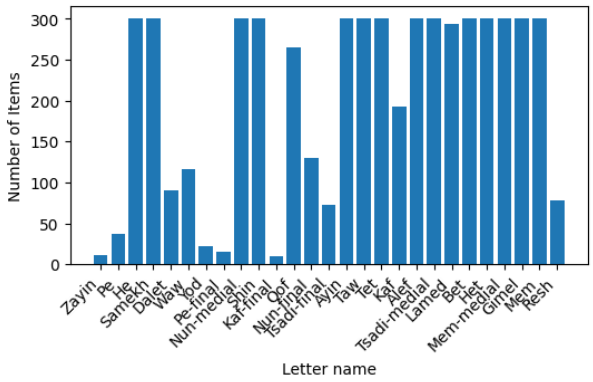


Figure 2.1: Number of observations per class

the model signals with a value of 1. This is also consistent with Tabik et al. (2017).

Important to note is that all except two scrolls are unlabeled; no ground truth is available. We will use the two scrolls that do have labels, as a validation set for our pipeline.

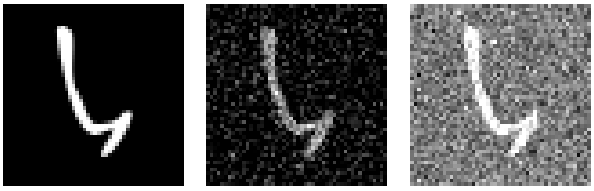
There is also a dataset provided called "Monkbrill", this dataset contains labeled images of the Hebrew letters sampled from scrolls. This set contains at maximum 300 observations per letter. However, as can be seen in Figure 2.1, there is a strong class imbalance. We also note that the images in this dataset are of varying size, with the height being between 27 and 77 pixels and the width between 17 and 196 pixels. The images are all scaled back to a size of 28 by 28 pixels, keeping the same aspect ratio and adding background padding where needed. When the images are downscaled, they are also transformed to a grayscale as this presevers more detail. Also, the images are inverted to have zero-valued background, just like the scrolls images.

2.2 Augmentation

For all augmentations, it holds that they are applied with probability p. This way, we can randomly combine augmentations effectively creating novel augmentation processes.

2.2.1 Random Gaussian Noise

For each input image, a noise image is created. This noise image consists of pixels that were randomly sampled from a Gaussian distribution, the mean and variance of which are supplied by the parameters. This is applied (added) to the original image. As this image may now have values outside the range [0, 1], we clip the image to range [0, inf]. Then, we normalize the clipped image to [0, 1]. An example transformation of the Random Gaussian Noise transformation can be seen in Figure Figure 2.2.



(a) Original **(b) Noise added** **(c) inverted, noise added, inverted back**

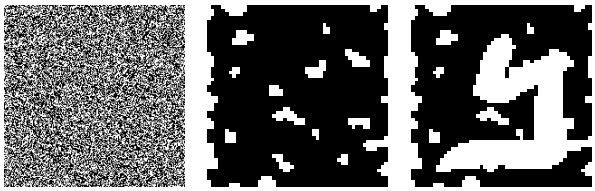
Figure 2.2: Adding Random Gaussian Noise

2.2.2 Random Blots

For each input image, a noise image of twice its height and width is created. The noise is then removed below a threshold of $0.5 + 0.01 \cdot$ blot_smallness. This parameter can be tuned to increase (smaller blot_smallness) or decrease blot size. The thresholded noise is then blurred using a Gaussian blur (with $\sigma X = \sigma Y = 5$). This results in rough-edged blots. These are then post-processed to look more like real-life inkblots. This includes removing tiny holes using CV2's open morph, removing tiny blots using CV2's close morph, and making blots rounder by applying a circle-shaped kernel. The blots are then applied using a bitwise-OR operation. The blots are always in the color of the background. An example of this transformation can be seen in Figure 2.3

2.2.3 Random Elastic Deformation

Each input is randomly deformed in the x-axis and y-axis. This is done by first generating 5 waves. The frequency is calculated using $\text{freq} = r \cdot \text{tuner}_w$. Where the width tuner is computed as .5% of the



(a) Generated noise **(b) Thresholded and blurred** **(c) Blots added to original**

Figure 2.3: Adding random blots. Note that this figure does not show all intermediate steps due to space constraints.

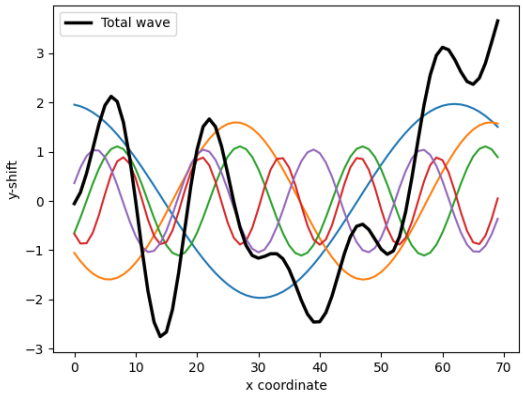


Figure 2.4: Five randomly generated waves and the result of combining them in bold black.

image width. The next width tuner is multiplied by a factor of 1.6 after generating a wave frequency. This way, N waves of different frequencies can easily be obtained. The r is sampled from a random distribution with a mean of 10 and a variance of 2. The amplitude is then calculated with using Equation 2.1.

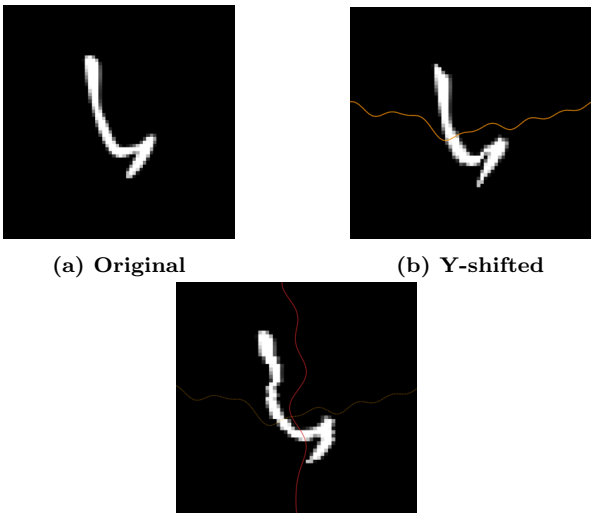
$$\text{amp} = \frac{0.2/W}{\sqrt{\text{freq}}} \tag{2.1}$$

It is important to note that each wave is shifted along the x-axis randomly. This ensures that the combination of waves is not too large at certain intervals.

In our implementation, we used $N = 5$ waves. An example of 5 generated waves and their combination can be found in Figure 2.4.

This wave is then used to shift all the pixels along the y-axis. Column i is shifted by a, where a is the value of the total x-wave at index i. Similarly, rows are shifted by the total y-wave. How much each pixel is shifted can finally be tuned using the strength reduction parameter, which divides the final amplitude of the total wave by a factor of strength reduction. This results in weaker elastic deformation.

For Hebrew character generation and augmentation of the training set, both an x-axis and y-axis transformation is applied using five waves each. No strength reduction is used. An example of this can be seen in Figure 2.5.



(a) Original **(b) Y-shifted**

(c) X-shifted

Figure 2.5: A generated image before deformation (a), after deforming on the x-axis (b), and after deforming on first the x-axis and then the y-axis (c).

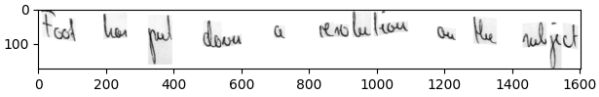


Figure 2.6: An example of a y-shifted sample from the IAM dataset.

For the IAM dataset, only a y-axis transformation is applied. The main reason for this is that an x-axis transformation already has the result of slightly tilting the letters. We concluded that an additional y-axis transformation would adds too little

variance without destroying the letters. A strength reduction of 2 was applied to the amplitude of the x-wave. An example of this can be seen in Figure 2.6.

2.3 Segmentation

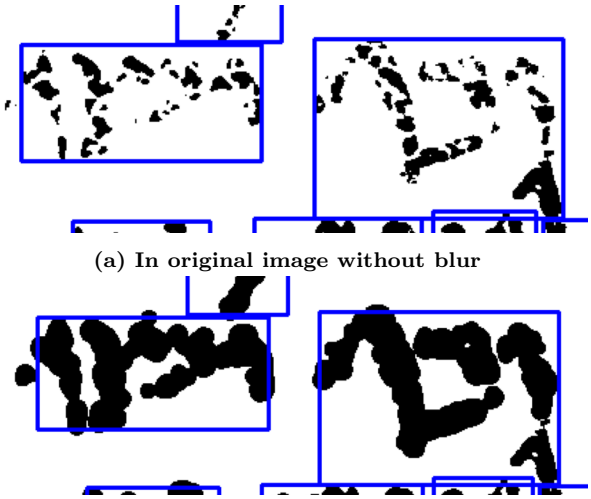
In order to be able to retrieve the individual letters within the image of the scroll we are first going to collect the connected components, which is a method inspired by Khorsheed (2002). A connected component is a set of pixels in which each pixel is either a direct neighbour of another pixel, or can be reached trough the direct neighbours of that pixel. Figure 2.7 gives an example of a connected component.

Figure 2.7: Example of connected components, the blue box contains one connected component.

It can happen, through deterioration of the handwritten text, that characters are fragmented into multiple small components while ideally they should be seen together as the same single connected component block. Detection of this kind is done by blurring small connected components with a Gaussian kernel and then rounding the pixel values to their binary values, such that small components can merge. The reasoning for this is that it's similar to morphological opening operations as described by Khorsheed (2002). Lastly another connected component analysis is performed on the small blurred components and larger regular components. The results of this can be found in Figure 2.8.

Whether a connected component is small enough to blur is dependent on the size of the character in the scroll. More precisely, an estimate of the size of a character is used for deciding which components to blur. First, we check if the mean of the height of the components is smaller than 1.5 times the median of the heights*. If this is the case then

*The height is used instead of the width because it has less variance. Characters are likely to connect horizontally, leading to wider components, and unlikely to connect vertically.



(a) In original image without blur

(b) In blurred image

Figure 2.8: Connected components found after blurring

the median is taken as an estimate of the size of a character is used. In the other case, the scroll contains many small components and the 90th percentile of the height of the components is used as an estimate. Connected components with a pixel count less than α times the estimate are blurred. A blur kernel size of β times the estimate is used, rounded up to an odd integer. Through visual inspection and grid search, values for α and β are set to $\alpha = 12.0$, $\beta = 0.3$. These and other hyperparameters relevant for this pipeline are also stated in Section 2.6.

From the blurred connected components, those that have few pixels in the un-blurred image are discarded. This threshold is $\alpha \cdot \gamma$ times the estimate of the size of a character. Again through visual inspection and grid search, $\gamma = 0.75$.

Note that blurring is only used to determine the bounding boxes of fragmented characters; further processing is done on the original image, using the bounding box found in the blurred image.

The issue now is that some bounding boxes clearly contain more than one letter. For the classifier to be able to work, we need to feed it individual letters. First, we need to know the size of a letter and use that to determine if we need to split some bounding boxes of the connected components into

multiple letters. For this we take the median of the height of the remaining bounding box, multiplied by parameter η. If a connected component is wider than this result, it is a candidate to be split. Parameter η was also varied in the grid search and set to $\eta = 1.5$

Next, these bigger bounding bounding need to be split into smaller boxes. This will be done using the vertical projection, a technique taken from Amin & Al-Fedaghi (1991), Amin (1989) and Amin (1988). We first project all pixel values on the x-axis. Then we take the minimum value and use this as the spot to split the image. Hence, we split the image on the spot where the stack of pixels is the lowest. To ensure that we do not split images close to the sides, we add a linear scaling term to the pixels on the side. Next to the added term, we also require the vertical projection to have peaks on both sides of the splitting point. This is to ensure we split at connections between characters and do not split longer characters themselves.

For both sides, the peak (maximum pixel value) times κ must be higher than the value where the split occurs. κ is the last parameter varied by grid search and is set to $\kappa = 0.5$.

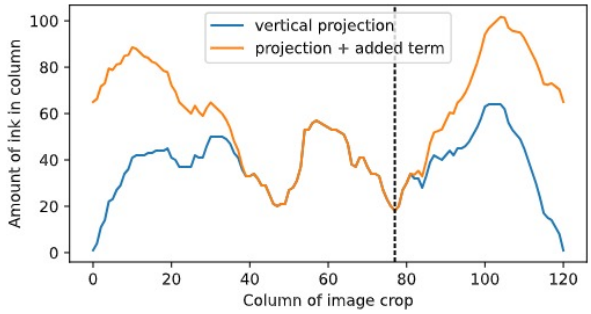


Figure 2.9: Vertical projection of the pixel values on the x-axis of Figure 2.7. The dotted line gives the split, as can be seen, the values at the sides are artificially increased.

Figure 2.10: Split character using the criterion in Figure 2.9.

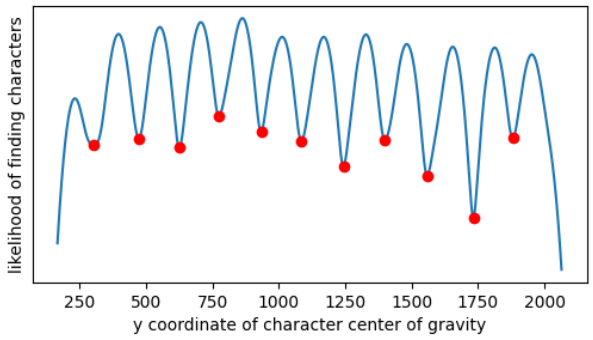
A result of the vertical projection is shown in Figure 2.9 and the associated split bounding boxes are shown in Figure 2.10.

Now that the connected components are split we are left with individual characters. These individual characters are then first used to do a line segmentation before being classified. This means that by using the center of gravity of the bounding boxes we make a horizontal projection on the y-axis. However, the number of characters per scroll is not that high, which would result in a very rough terrain with a lot of local minima, making it hard to split the lines. This is solved by using a tool from statistics called kernel density estimation. Instead of projecting the centers of gravity directly on the y-axis, a Gaussian distribution is plotted with $\mathcal{N}(\mu, \sigma)$ where μ is the y coordinate of the center of gravity. The $\sigma = 25$ is a hyperparameter found by manual testing on the unlabeled train dataset. The result of this is a sum of normal distributions. We find the local minima of the sum of distributions and take those as the boundaries of different lines of texts. Then we assign each character to a line of text based on which line their center of gravity lies. The line segmentation result of one scroll can be seen in Figure 2.11.

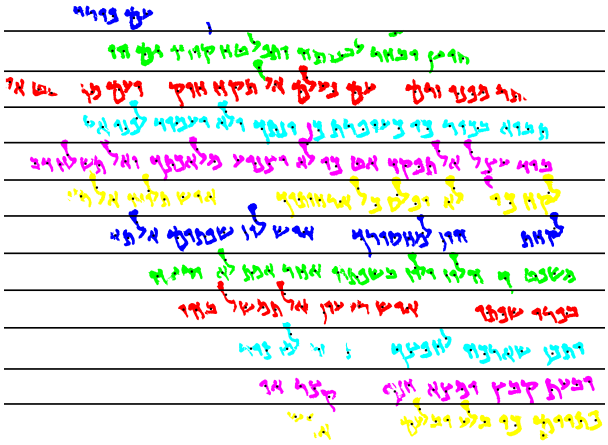
2.4 Classification

So far, we have segmented characters and also assigned them to a line. These segmented characters now need to be classified. This leads to a classification problem with 27 classes. This is a task similar to the digit classification task on the popular MNIST dataset. Therefore, we take inspiration from Tabik et al. (2017), and we use a convolutional neural network of which the structure is presented in Figure 2.12. Whereas a loss function we use the cross-entropy loss, since we are dealing with a multi-class classification problem. The model is optimized using Stochastic Gradient Descent (SGD) as was also done in Tabik et al. (2017). The hyperparameters used for SGD and other hyperparameters for the current pipeline are stated in Section 2.6.

To improve convergence and decrease the risk of overfitting, batch normalization Ioffe & Szegedy (2015) is applied after every convolutional layer. To further regularize the network, dropout Srivastava et al. (2014) is used after the first two fully con-



(a) Kernel density estimation with the center of gravity of the segmented characters. Red dots signify the boundaries between different text lines.



(b) Visual of the associated line segmentation. Horizontal lines are the boundaries between text lines. Small black dots are the centers of gravity.

Figure 2.11: Line segmentation

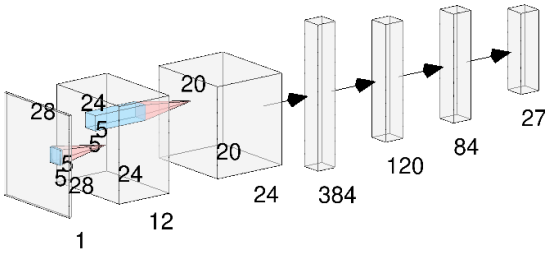


Figure 2.12: A representation of the CNN structure. The model consists of three convolutional layers with kernel size 5, going from 1 to 12 to 24 channels, followed by three fully connected layers.

nected layers during training.

As mentioned earlier and shown in 2.1, the data is quite unbalanced. In order to still allow the model to sufficiently learn information from the less-frequent classes, data is sampled using weighted sampling during training. Every datapoint gets assigned a weight inversely proportional to the class count of that datapoints class. Batches are then sampled with replacement, with the weight determining the probability for a datapoint to be selected. Weighted sampling was only used for sampling training data, validation data was sampled uniformly without replacement every epoch.

We implemented augmentation techniques from Section 2.2 in order to increase the size of our training dataset. The goal of this is to increase the performance on the smallest classes by introducing more samples of those classes, as well as increasing overall model robustness and generalizability by introducing more variance in the dataset.

Moreover, we implemented a random crop from the Torchvision module, where the output image is of size (20,20).

Before training on the IAM data, the model is pre-trained using a synthetic dataset which is generated by applying the data augmentations in the same manner as on the regular dataset, but then on the Habbakuk font. The decision to include pre-training was made based on the small sample size of the Monkbrill dataset as well as the class imbalance, with the goal of improving performance. Every epoch, a single batch of 16 examples is generated for training. The model is pre-trained for 5000 epochs. Every 100 epochs, the performance is measured using the Monkbrill dataset. The results of which are presented in Section 4.

In order to assess model performance, the model was trained using k-folds with 5 folds for 100 epochs each. The resulting performances were used to experiment with hyperparameter choices in order to improve them. The mean validation results from the k-folds training process with our final hyperparameter settings will be presented in Section 4. However, the final model selected for the working phase was trained on the complete dataset in order to fully utilize the data. The final model was also trained for 100 epochs. This training run had no validation data, therefore no results will be reported for it.

2.5 Pipeline evaluation

The hyperparameters for classification and segmentation are summarized in Table 2.2 and Table 2.1.

The Levenshtein distance (Levenshtein, 1966) will be used for evaluating the complete pipeline. This is a metric for comparing strings of text. The distance represents the minimum number of character insertions, deletions, or replacements needed to transform one string into the other.

In the ground truth of the two labeled scrolls, there are multiple lines of Hebrew text where each character can be divided into one of three classes: A normal character, a character that has some deterioration and is hard to classify, or a character that has deteriorated a lot and is very hard to classify. The output of the pipeline simply produces characters without a notion of how deteriorated they are. The output is evaluated in the strictest sense when compared to the ground truth, so even if a difficult-to-classify character is not present in the output it still adds to the distance.

The evaluation is done line by line. The distance is calculated between the first line of the ground truth and the first line of the pipeline output, then both texts proceed to the next line iteratively until both texts finish their last lines[†]. The distances for every line pair are summed together to give the final Levenshtein distance on the whole output.

2.6 Hyperparameters

This section holds a small overview of the hyperparameters that were used in the Hebrew character pipeline. We chose to gather all hyperparameters in this subsection to improve the readability of the already-complicated pipeline.

Table 2.1 shows the hyperparameter values used during the segmentation process. Table 2.2 shows the hyperparameter values used for the classification model.

α	β	γ	η	κ
12.0	0.3	0.75	1.5	0.5

Table 2.1: Resulting segmentation hyperparameters after grid search.

[†]If the number of lines differs in the two texts, then empty strings will be paired with the remaining lines.

Batches	5000
Epochs	100
Batch size	16
Optimizer	SGD
Learning rate (SGD)	1e-3
Momentum (SGD)	.9

Table 2.2: Hyperparameters for the pre-training and training phase. The number of batches indicates the number of generated batches that was used for pre-training. The pre-training hyperparameters are the same as the training hyperparameters.

3 Methods - IAM

Secondly, we are also tasked with handwritten text recognition on the IAM dataset. We again will set up an end-to-end pipeline that is able to read text from images and write it to a file. Let us first discuss the data description after which we will present the algorithms used.

Parts of the implementations of these methods were realized with the help of ChatGPT OpenAI (2023).

3.1 Data description

The IAM Handwriting Database (Marti & Bunke, 1999) is a collection of handwritten English text used for training and evaluating handwriting recognition systems. Specifically, in this research, we are going to use a subset of dataset, consisting of labeled line text images, which are images of handwritten text lines. A subset of this dataset is preserved by the lectures for testing purposes.

To improve the model's performance, we have also gathered two additional datasets. First, the LAM dataset (Cascianelli et al., 2022) is a large labeled line-level handwriting recognition dataset of Italian ancient manuscripts, consisting of 25.823 observations. Just like IAM, this dataset consists of already line-segmented images. However, the language of course is different.

Secondly, we will use the dataset from Kaggle (Gautam, 2020) consisting of English-labeled text in the same style as IAM. This dataset consists 396 observations.

3.2 The Pipeline

Let us now describe the pipeline. For this pipeline, a model based on TrOCR (Li et al., 2023) is used. As described in the introduction, TrOCR is a Transformer based Optical Character Recognition (OCR) model consists of an encoder and a decoder. In all versions of the pipeline, the base TrOCR model parameterss are used as a starting point. The encoder is initialized with the weights of Bao et al. (2022),and the decoder layer is initialized with the weights of Liu et al. (2019). This model is downloaded from Li et al. (2021).

The model will be trained using several different processes, of which thebest-performingg model is selected for the working phase. The first model is a baseline model that is only trained on the IAM dataset, with no further adaptations. There are also models trained that are variations on thisbaselinee, which will be presented next.

A first variation on thebaselinee model is to introduce pre-training. The model will be pre-trained on the LAM dataset and the Kaggle dataset, as introduced above. After pre-training the model will be fine-tuned on the IAM dataset. Since we expect the distribution of the Kaggle dataset to be closer to that of the LAM dataset, we will first pre-train on the LAM dataset and then we will pre-train on the Kaggle dataset.

We also inspect the effect of both dropout and data augmentations. Important to note is that the use of dropout or data augmentation is consistent between pre-training and fine-tuning. So for example if dropout is used during pre-training it is also used during fine-tuning.

The idea of dropout is that during training some of the neurons do not participate in the network. The goal of this is to improve generalization and decrease overfitting (Srivastava et al., 2014). Since a pre-trained model is used, we are limited by the implementation of a third party. This means that in this case, dropout is only applied to the visual encoder model. There, dropout is applied to all attention and output layers with a rate of .1.

Data augmentation is applied as described in Section 2.2. The probabilities used for augmentation are the same for the pre-training phase as for the training phase. The hyperparameters for the augmentations are as follows:

- Random Elastic deformation: probability = 0.5, strength reduction = 2
- Random blots: probability = 0.5, blot smallness = 3, elipse kernel shape = (9,9).
- Random gaussian noise: probability = 0.9, mean = 0.5, variance = 0.5, double invert = True.

The above presented techniques are used in combination and we apply different pre-training settings. The following pre-training settings are tested:

1. Pre-training only (Pr)
2. Pre-training and data augmentation(PrAug)
3. Pre-training, data augmentation, and dropout (PrAugDrpt).

The fine-tuning counterparts have similar names. The baseline model without pre-training will be referred to as Ft. Continuing onwards, we have FtPr for the fine-tuned model on the pre-trained model, etc.

Important to note is that the pre-training of the models is done for a fixed number of epochs, namely 10 and 20, respectively for the LAM and the Kaggle dataset. After each epoch, the model is saved if it has a lower validation loss. The model with the lowest validation loss proceeds to the next step. Hence, after pre-training on LAM, the model with the lowest validation loss is pre-trained on the Kaggle dataset. The model that then again has the lowest validation loss is fine-tuned on IAM. Since we use the validation loss, we do not have to worry about overfitting. This will be done for *each* pre-train setting. Hence after pre-training, there are three models to be fine-tuned.

During the pre-training the cross-categorical loss is optimized and the batch size is set to 20. AdamW (Loshchilov & Hutter, 2019) is used as an optimizer with a learning rate of $5e-5$.

To further prevent overfitting we apply L_2 regularization with a strength of 10^{-2}. These hyperparameters were suggested by (Li et al., 2023).

After pre-training, the models are all being fine-tuned using a train/validation split. The performance on the validation set will be used to compare the models. The models will be trained for a fixed number of 20 epochs. For each setting, the validation loss will be recorded and the model with the lowest validation loss will be stored. Between these models, we will choose the model that has the low-

est loss to deploy as the model that is used in the pipeline.

The performance of the models will be assessed using both the Word Error (WER) and the Character Error (CER). Note that a word is incorrect if one of the letters within that word is not correct and the CER is a "per character" error. Therefore, one would expect WER>CER.

4 Results

The results of both pipelines are described in this section. First, the Hebrew pipeline will be discussed, and after that the IAM pipeline.

4.1 Hebrew character pipeline

The results of the segmentation grid search can be found in Appendix A. Furthermore, there are some more segmentation visuals of different scrolls in Appendix B.

Furthermore, in the next sections about the Hebrew pipeline we report the results of the pre-training, the training on handwritten characters, and lastly the evaluation of the two labeled test scrolls.

4.1.1 Pre-training results

In Figure 4.1 the training and validation loss are shown for the pre-training on Hebrew text font. The accuracy of this pre-training is shown in Figure 4.2

We see that the training loss goes down and converges quite quickly, but the validation loss stays high and never comes close to a low loss value. The accuracy does have a general trend of improving, going from about 12% accuracy at the start to about 24% after 5000 epochs.

4.1.2 K-fold results

Next are the classification results on handwritten Hebrew characters after the pre-training on Hebrew text font, also called fine-tuning in this section. In Figure 4.3a the training loss and validation loss are shown. In Figure 4.3b the accuracy is shown.

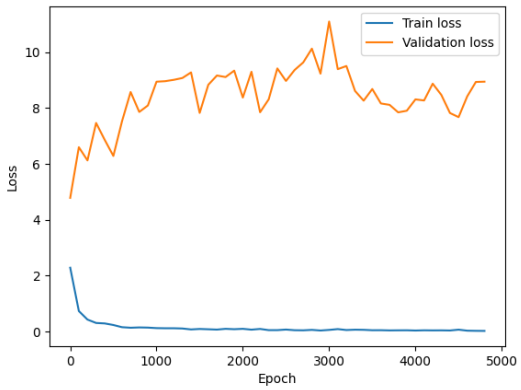
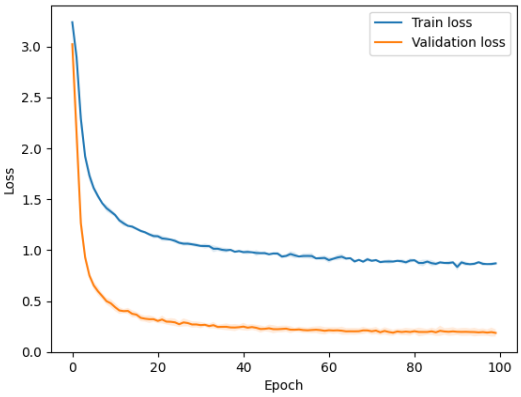
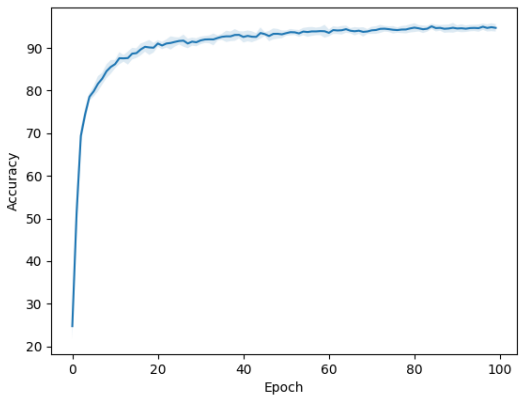


Figure 4.1: Train and validation loss of pre-training process over 5000 epochs. Validation loss is measured over the complete Monkbrill dataset every 100 epochs.



(a) Mean train and validation loss of fine-tuning process over 5 folds of 100 epochs each.



(b) Mean accuracy of fine-tuning process over 5 folds of 100 epochs each.

Figure 4.3: Hebrew fine-tuning

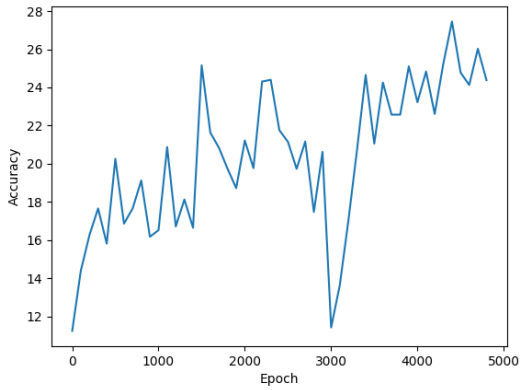


Figure 4.2: Accuracy over 5000 epochs during pre-training. Accuracy is measured over the complete Monkbrill dataset every 100 epochs.

Both the training loss and the validation loss go down and converge. What is notable is that the validation loss is lower than the training loss. One should keep in mind that during training image augmentation was used, while it was not used during validation.

The accuracy starts at about 25% and rises quickly. The final accuracy score reaches a mean of 94.73 with a standard deviation of 0.87 across the 5 folds. The average confusion matrix is presented in Figure 4.4. Overall the performance is as expected, where some smaller classes are harder to classify.

4.1.3 Evaluation

In the final pipeline, the Levenshtein distance for the two labeled scrolls are 83 and 224. The names of these scrolls are '25Fg-001' and '124-Fg004' respectively, with total character lengths of 138 and 350. When comparing the output of our pipeline to an empty string, the distance gets reduced by 39.5% and 36.0% respectively.

4.2 IAM pipeline

In Table 4.1 the character error rate and the word error rate from the pre-training are reported on the LAM dataset and the Kaggle dataset. More extensive plots of the training across epochs can be found in Appendix C.

	LAM		Kaggle	
	CER	WER	CER	WER
Pr	9.89	27.34	20.97	35.01
PrAug	10.04	26.94	24.63	40.96
PrAugDrpt	13.18	32.20	26.87	40.57

Table 4.1: CER and WER scores of the denoted pre-training processes. The scores shown are the scores at the time when validation loss on the dataset that is being used for training is the lowest.

The character error rate and the word error rate for the LAM dataset is lower than the Kaggle one for every model. Furthermore, the model with only pre-training performs best, except for the word error rate on the LAM dataset where the pre-training with augmentation performs slightly better.

Lastly, the character error rate and the word error rate for the final fine-tuning models on IAM are reported in Table 4.2.

	IAM	
	CER	WER
Ft	9.87	17.61
FtPr	13.54	24.56
FtPrAug	12.11	22.37
FtPrAugDrpt	13.75	25.87

Table 4.2: CER and WER scores of the denoted training processes. The scores shown are the scores at the time when validation loss is lowest.

What can be noted is that the model without pre-training, augmentation, and dropout (Ft) performs best. It achieves a character error rate of 9.87% and a word error rate of 17.61%. Because of its performance, this is the final model chosen for the working phase.

References

Amin, A. (1988). Ocr of arabic texts. In *Pattern recognition.*

Amin, A. (1989). Machine recognition and correction of printed arabic text. *IEEE Transactions on Systems, Man, and Cybernetics*.

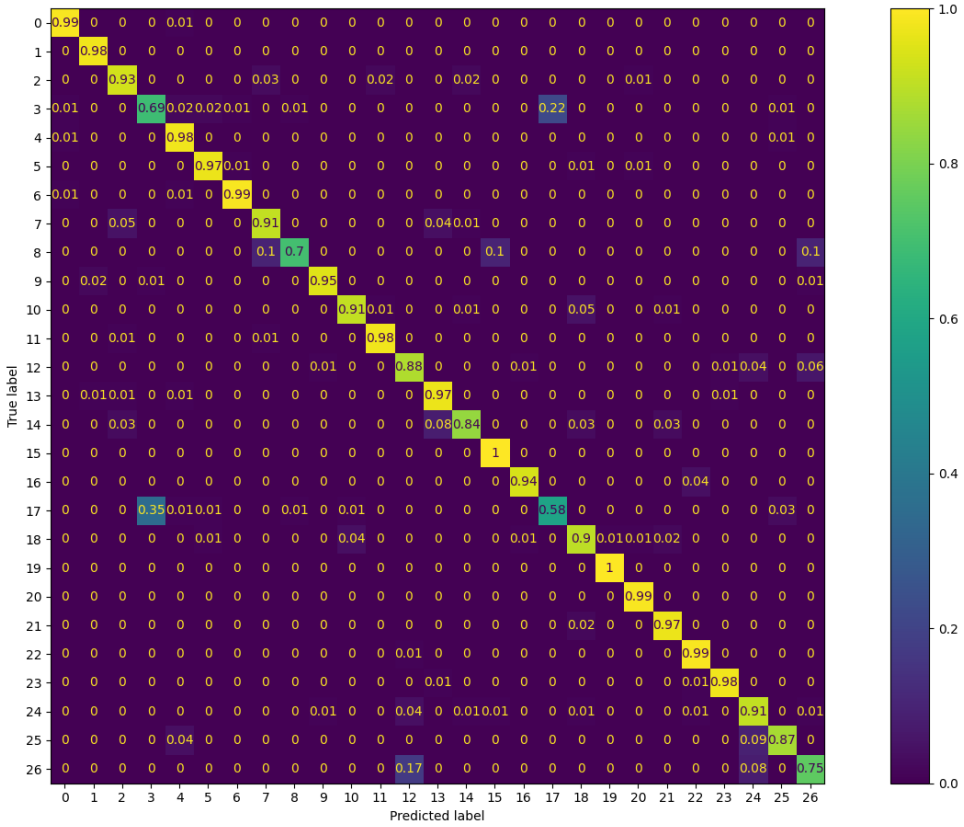


Figure 4.4: Confusion matrix of the classification model used in the Hebrew pipeline.

Amin, A., & Al-Fedaghi, S. (1991). Machine recognition of printed arabic text utilizing natural language morphology. *International Journal of Man-Machine Studies*, *35*(6), 769-788.

Bao, H., Dong, L., Piao, S., & Wei, F. (2022). *Beit: Bert pre-training of image transformers.*

Cascianelli, S., Pippi, V., Maarand, M., Cornia, M., Baraldi, L., Kermorvant, C., & Cucchiara, R. (2022). *The lam dataset: A novel benchmark for line-level handwritten text recognition.*

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255).

Gautam, S. (2020). *English handwritten line dataset.* Retrieved 18-06-2024, from `https://www.kaggle.com/datasets/sushant097/english-handwritten-line-dataset?resource=download`

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep residual learning for image recognition.*

Iceland, M., & Kanan, C. (2023). *Understanding the benefits of image augmentations.*

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456).

Khorsheed, M. S. (2002). Off-line arabic character recognition–a review. *Pattern analysis & applications*, *5*, 31–45.

Levenshtein, V. I. (1966, February). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, *10*, 707.

Li, M., Lv, T., Chen, J., Cui, L., Lu, Y., Florencio, D., ... Wei, F. (2023). Trocr: Transformer-based optical character recognition with pre-trained models. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 37, pp. 13094–13102).

Li, M., Lv, T., Cui, L., Lu, Y., Florencio, D., Zhang, C., ... Wei, F. (2021). *TrOCR (base-sized model, pre-trained only).* Retrieved from `https://huggingface.co/microsoft/trocr-base-stage1`

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). *Roberta: A robustly optimized bert pretraining approach.*

Loshchilov, I., & Hutter, F. (2019). *Decoupled weight decay regularization.*

Marti, U.-V., & Bunke, H. (1999). A full english sentence database for off-line handwriting recognition. In *Proceedings of the fifth international conference on document analysis and recognition. icdar'99 (cat. no. pr00318)* (pp. 705–708).

OpenAI. (2023). *Chatgpt: A conversational agent.* `https://www.openai.com/chatgpt`. (Accessed: 2024-06-18)

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Tabik, S., Peralta, D., Herrera-Poyatos, A., & Herrera, F. (2017). A snapshot of image preprocessing for convolutional neural networks: case study of mnist. *International Journal of Computational Intelligence Systems*, *10*(1), 555–568.

Tobing, T., Yildirim Yayilgan, S., George, S., & Elgvin, T. (2022, 06). Isolated handwritten character recognition of ancient hebrew manuscripts. *Archiving Conference*, *19*, 35-39. doi: 10.2352/issn.2168-3204.2022.19.1.8

Wise, M., Abegg, M., & Cook, E. (1996). *The dead sea scrolls.* San Francisco.

Individual contributions

Rob Primarily focused on the segmentation methods in the Hebrew pipeline (components, characters and lines). Visualization of the scrolls. Contributions to Hebrew pipeline methods and results sections of the report.

Thijs Focused primarily on augmentation methods, dataset generation, model pretraining and some algorithmic pieces in other parts of the pipeline such as gathering the connected components of fragmented letters.

Otto Focused on initial CNN network and architecture. Small help with segmentation and gridsearch. Connecting individual processes in the pipeline. Relatively a lot of time in to the report.

Roben Worked mainly on the classifier part of the Hebrew pipeline as well as the training process of the IAM pipeline. Further contributed to the results as well as the methods of both pipelines.

A Segmentation grid search

α	β	γ	η	κ	distance
18.0	0.2	1.0	1.5	0.3	77
18.0	0.2	1.0	1.5	0.5	78
18.0	0.3	0.75	1.5	0.3	78
18.0	0.2	1.0	1.5	0.8	78
12.0	0.45	1.0	1.5	0.3	79

Table A.1: Top 5 results in segmentation grid search. Scroll 25Fg-001

α	β	γ	η	κ	distance
8.0	0.3	0.75	1.0	0.5	203
8.0	0.2	0.6	1.0	0.3	205
8.0	0.3	1.0	1.0	0.5	206
8.0	0.2	0.75	1.0	0.3	206
8.0	0.45	0.75	1.0	0.5	207

Table A.2: Top 5 results in segmentation grid search. Scroll 124-Fg004

In the Hebrew character pipeline a grid search was performed on the segmentation parameters. The two labeled scrolls were used for validation with the Levenshtein distance.

The five parameters were first set using visual inspection on all the labeled and unlabeled scrolls. Then in the grid search each parameter was also tested for a reasonable higher and a reasonable lower value. Those values are $\alpha = \{8.0, 12.0, 18.0\}$, $\beta = \{0.2, 0.3, 0.45\}$, $\gamma = \{0.6, 0.75, 1.0\}$, $\eta = \{1.0, 1.5, 2.25\}$, $\kappa = \{0.3, 0.5, 0.8\}$.

The top 5 results for both scrolls are shown in Table A.1 and Table A.2. There is no unanimous vote for the best parameters. For example we see that one scroll performs better with high values of α and the other scroll performs better with low values of α (As a reminder, higher values of α will result in more blurred connected components). It is therefore difficult to choose the best parameters for general performance based on only two labeled scrolls. In the end, parameters that lie in the middle were chosen as a compromise between the two scrolls and based on the initial visual inspection.

B Some character and line segmentation visuals

Three scrolls were randomly chosen from which to show the character segmentation and line segmentation results.

Figure B.1: Scroll P168-Fg016-R-C01-R01

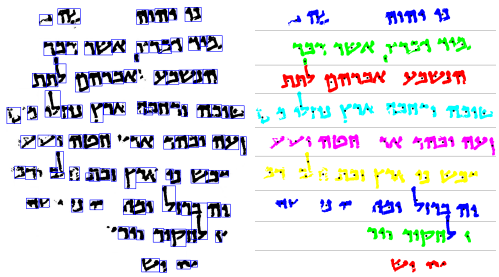


Figure B.2: Scroll P166-Fg007-R-C01-R01

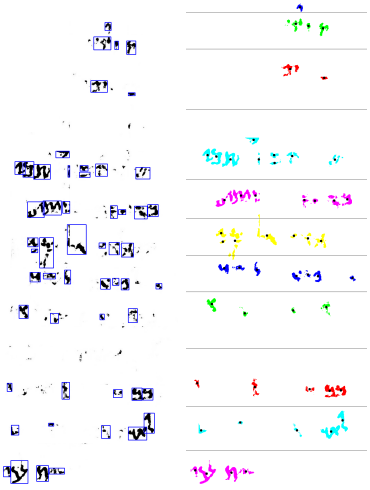
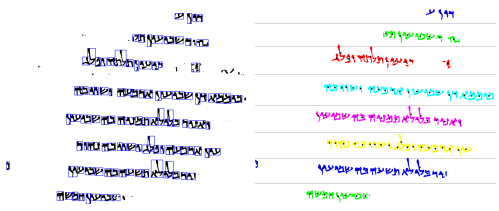
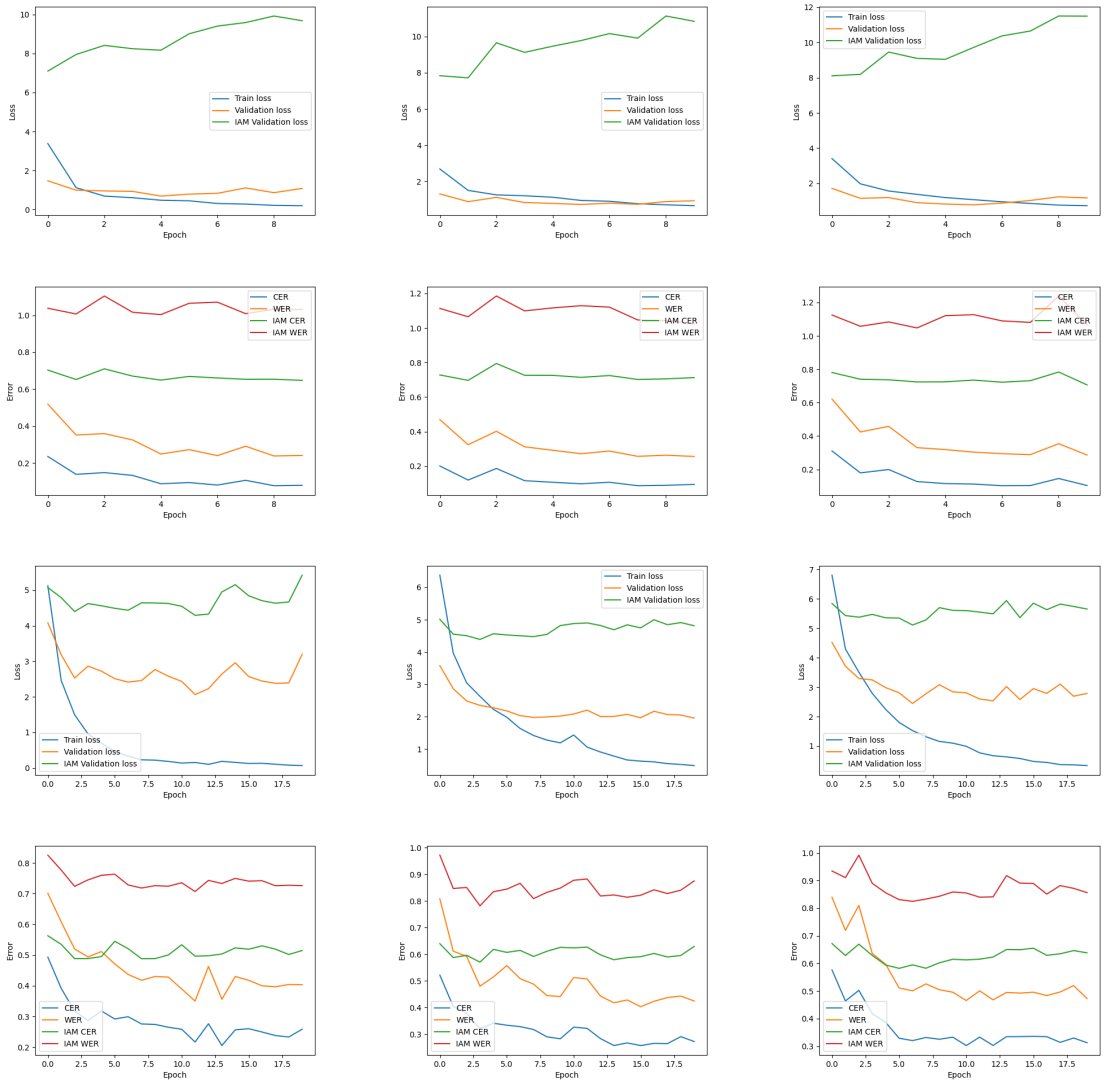


Figure B.3: Scroll P846-Fg001-R-C01-R01



C IAM plots

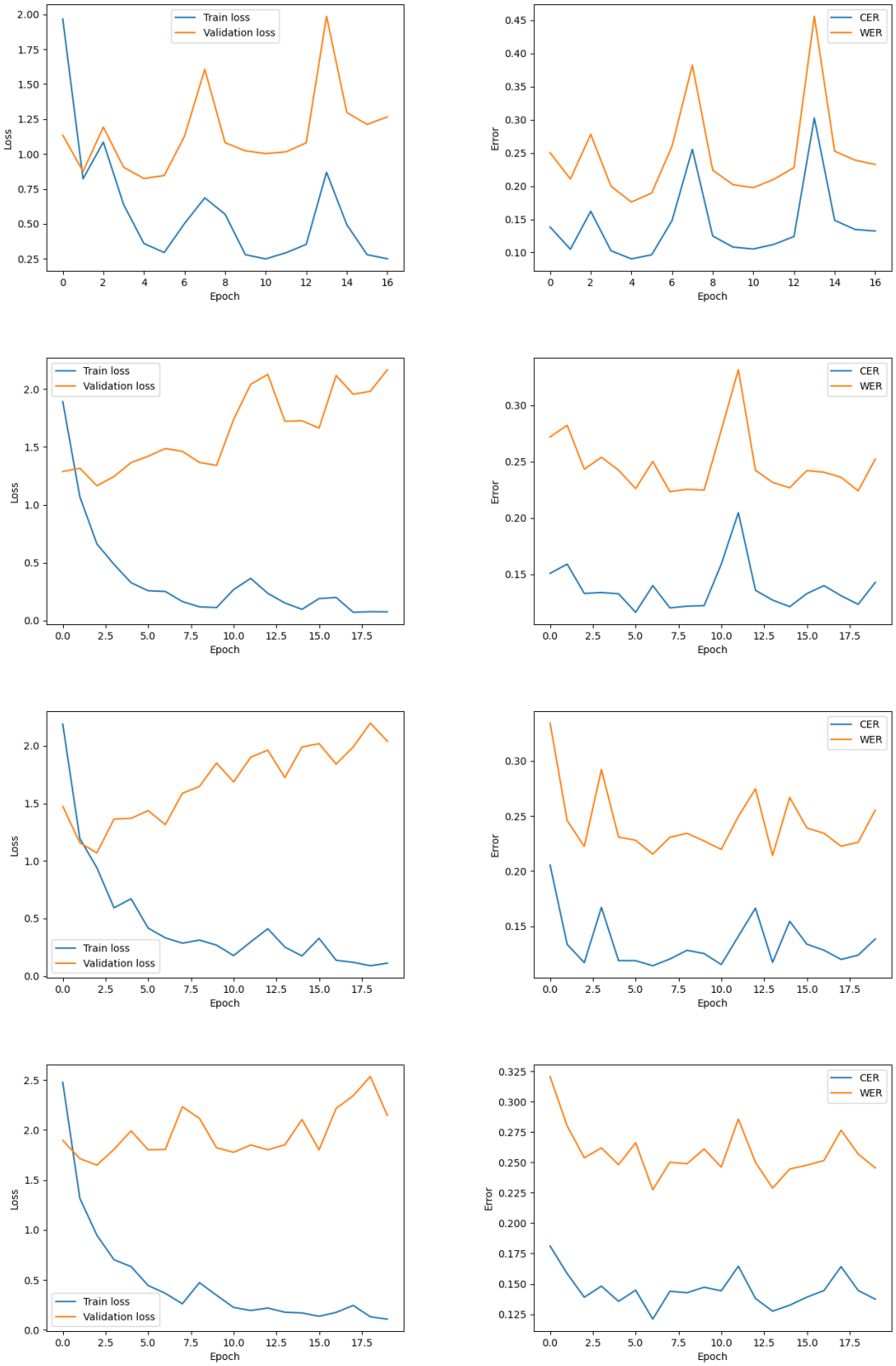


(a) No augmentations or dropout. (Pr from Section 3)

(b) With augmentations, no dropout. (PrAug from Section 3)

(c) With augmentations and dropout. (PrAugDrpt from Section 3)

Figure C.1: Results of pre-training for the IAM line recognition task. Subcaptions denote the training settings for that column. The rows represent the following results, from top to bottom: Loss during pre-training on LAM, error during pre-training on LAM, loss during pre-training on kaggle, error during pre-training on self-made.



(a) Train and validation loss during training.

(b) Error rates during training.

Figure C.2: Results of fine-tuning on the IAM dataset. The type op plots is specified per column in the subcaption. From rows represent the following training settings (label in parentheses denotes the label specified in Section 3), from top to bottom: No pre-training, no data augmentation and no dropout (Ft); Pre-trained model, with data augmentations, no dropout (FtPrAug); Pre-trained model, with data augmentation and dropout (FtPrAugDrpt).