



UPPSALA  
UNIVERSITET

UPTEC IT 23002  
Examensarbete 30 hp  
Januari 2023

# Handwritten Text Recognition Using a Vision Transformer

---

Jonathan Kurén and Martin Sundberg







UPPSALA  
UNIVERSITET

## Handwritten Text Recognition Using a Vision Transformer

---

Jonathan Kurén and Martin Sundberg

### **Abstract**

The aim of this project is to create a method for offline handwritten text recognition using a vision transformer. It consists of two parts, where the first one segments all words in a document into separate images and the second one which recognizes the word on each image. The focus in this project will be on the second part which will use a Vision Transformer. The Vision Transformer was evaluated on the IAM dataset and compared with the state-of-the-art. The results are promising but there is still some work to be done until the performance is equal to the state-of-the-art. An ablation study was done to show the effectiveness of different methods used to improve the result. There was also an error analysis done to show how the model performed on different letters and a furthermore a visual analysis was done to highlight common errors.

**Teknisk-naturvetenskapliga fakulteten**

**Uppsala universitet, Utgivningsort Uppsala/Visby**

Handledare: Anders Hast Ämnesgranskare: Filip Malmberg

Examinator: Lars-Åke Nordén



## Sammanfattning

Syftet med detta projekt är att skapa en metod för offline handskriftsigenkänning som använder en vision transformer. Den består av två delar, där den första segmenterar alla ord i ett dokument till separata bilder och den andra känner igen ordet på varje bild. Fokus i detta projekt kommer att ligga på den andra delen som kommer att använda en Vision Transformer. Vision Transformern utvärderades på IAM-datasetet och jämfördes med den senaste tekniken. Resultaten är lovande men det återstår fortfarande en del arbete att göra tills prestandan är lika med den senaste tekniken. En ablationsstudie gjordes för att visa effektiviteten av olika metoder som används för att förbättra resultatet. Det gjordes även en felanalys för att visa hur modellen presterade på olika bokstäver och en visuell analys gjordes även för att belysa vanliga fel.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Aim . . . . .	1
1.2	Challenges and Limitations . . . . .	2
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	Handwritten Text Recognition . . . . .	2
2.2	Neural Networks . . . . .	2
2.3	Convolutional Neural Networks . . . . .	4
2.4	Performance Metrics . . . . .	6
2.4.1	HTR metrics . . . . .	6
2.4.2	Classification metrics . . . . .	6
2.5	Handwriting . . . . .	7
2.6	Tokenization . . . . .	7
<b>3</b>	<b>Related Work</b>	<b>7</b>
3.1	Transkribus . . . . .	7
3.2	Robust Handwriting Recognition with Limited and Noisy Data . . . . .	7
3.3	Fully Convolutional Networks for Handwriting Recognition . . . . .	8
3.4	Scan, Attend and Read: End-to-End Handwritten Paragraph Recognition with MDLSTM Attention . . . . .	9
3.5	Boosting the Deep Multidimensional Long-Short-Term Memory Network for Handwritten Recognition Systems . . . . .	9
3.6	Transformers in Computer Vision . . . . .	10
<b>4</b>	<b>Methods</b>	<b>10</b>

4.1	Segmentation . . . . .	10
4.2	Data Preprocessing . . . . .	12
4.2.1	Deslanting . . . . .	12
4.2.2	Grayscale . . . . .	12
4.2.3	Image Size . . . . .	13
4.3	Data Generation . . . . .	13
4.4	Vision Transformer . . . . .	15
<b>5</b>	<b>Experiments</b>	<b>17</b>
5.1	Datasets . . . . .	17
5.2	Model Training . . . . .	18
5.3	Hyperparameters . . . . .	18
<b>6</b>	<b>Handwritten Text Recognition Results</b>	<b>19</b>
<b>7</b>	<b>Complete System</b>	<b>20</b>
<b>8</b>	<b>Discussion</b>	<b>23</b>
8.1	Ablation Study . . . . .	23
8.2	Error analysis for the Vision Transformer . . . . .	24
8.3	Visual Analysis of Images . . . . .	26
8.4	Analysis of the Complete System . . . . .	28
<b>9</b>	<b>Future Work</b>	<b>29</b>
<b>10</b>	<b>Conclusion</b>	<b>30</b>
<b>A</b>	<b>Images</b>	<b>35</b>





# 1 Introduction

The handwritten documentation of information has been and still is an important way of recording and preserving important knowledge for the future. A problem with handwritten documents is that they are easily damaged or destroyed. The ability to digitally record handwritten documents can be crucial in order to save the documents for the future, and by extracting the text from the documents and saving it the documents can be made searchable. Since the handwriting of different persons can vary substantially this is a complicated problem, which could be solved using handwritten text recognition(HTR).

HTR is used to take an image of handwritten text and letting a machine read and recognize the text. The system needs to be able to separate the words and read them in the correct order before outputting the predicted text. The research on HTR is ongoing and new methods are being developed. The solutions can be made in multiple ways, for example with different combinations of neural networks.

Recently transformers have been applied to image analysis as vision transformers. They have been proven efficient in image classification where the vision transformer was able to perform as well as the state-of-the-art convolutional neural networks. This project will investigate the use of vision transformers in handwritten text recognition.

## 1.1 Aim

The aim of this project is to create a method for offline handwritten text recognition which could be used on handwritten documents. There are two parts which goes into a HTR method, the first one is segmenting all words in a document and the second one is to recognize what each word is. In this project there will be a bigger focus on the recognition part where the goal is to use a Vision Transformer as a model, meanwhile for the segmentation part an existing method will be used. In the end, the goal is to have a method which takes a handwritten document as input and outputs the text digitally. This should be done regardless of language as long as the characters are a part of the Latin alphabet or Arabic numerals.

The questions this project aims to answer are the following:

- Is it possible to create a model used for HTR using a Vision Transformer?
- Can the model challenge or outperform already existing models?
- Can the model be used for real world problems?

## 1.2 Challenges and Limitations

One of the biggest challenges is to create a dataset of labelled handwritten text that could be used to train the ML model. If it is not possible to create a big enough dataset, transfer learning would have to be used.

The segmentation part will be done using an existing method since the focus will be on the recognition part. This will lead to the segmentation being a possible bottle neck where if the performance is not good enough, there will be no additional work done to improve the performance.

Another key challenge is to create a HTR method which can recognize words it has not trained on. This is important if the method is to work with different languages, since almost all publicly available datasets are in English. The system will be limited to character and numbers from the Latin alphabet and Arabic numerals and will not be able to find commas, punctuations and newlines.

## 2 Background

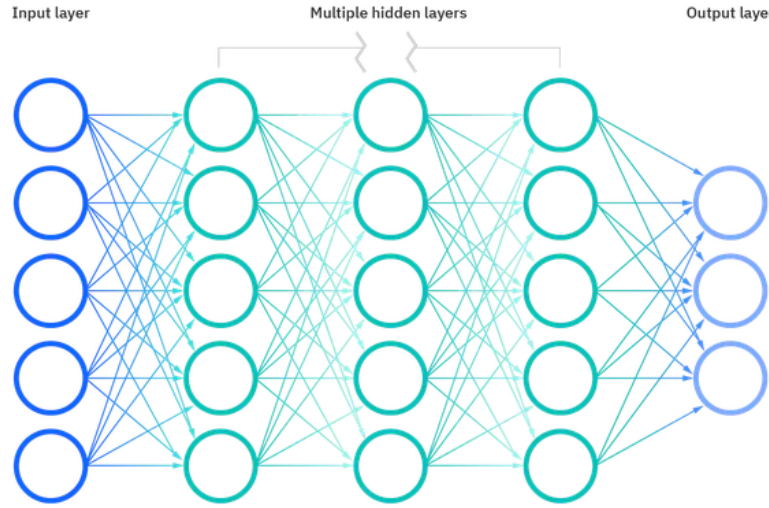
### 2.1 Handwritten Text Recognition

There are two types of handwritten text recognition, offline and online. Offline is when the only available information is the completed writing as an image. Online is where the writing is stored as coordinates in a two-dimensional plane as a function of time in the order the points were written. This captures the order the strokes were done in which could be utilized in the HTR. However, this also requires a special electronic pen that saves this information. Online HTR is generally easier than offline HTR and the data requirement is also much less, where a word in online HTR is a couple of hundred bytes and a word in offline HTR is a couple of kilobytes[18].

### 2.2 Neural Networks

Neural networks, also called artificial neural networks (ANNs), is a subset of machine learning [2]. A neural network is made up of at least an input layer, a hidden layer and an output layer. This can be seen in Figure 1. The number of hidden layers can vary

greatly, but if there are more than 3 layers in total, including the input and output layer, the network can be called a deep learning network.



**Figure 1** Image showing the structure of a neural network

A neural network works by having a node in a layer being connected to all other nodes in the next layer, this is called a fully connected layer. Each connection has an associated weight and threshold. The value of a node is determined by taking the input value and multiplying it with the weight. This new value is then passed through an activation function and that value is then used as the output of the node. There are many different activation functions such as:

$$RELU : \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

$$Sigmoid : \frac{1}{1 + e^{-x}}$$

$$Binary : \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

and depending on which one is used the performance of the neural network will differ. Therefore it is good practice to test multiple ones to know which is best.

When training a neural network the goal is to minimize a loss function. This could

for example be the mean squared error where  $Y_i$  is target and  $\hat{Y}_i$  is the output of the neural network:

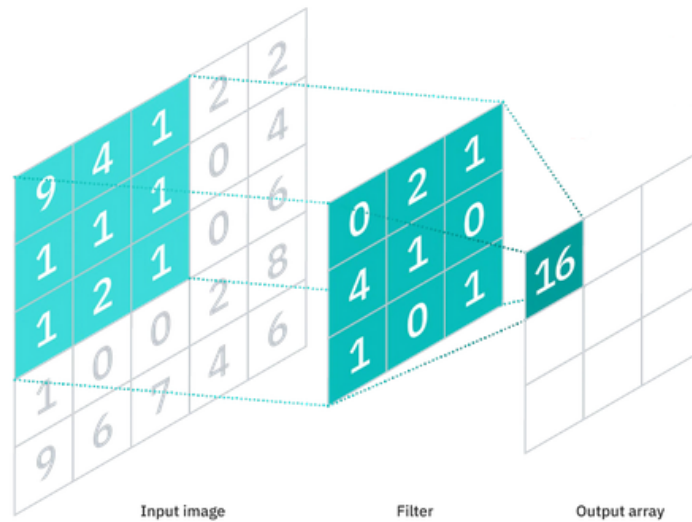
$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

The weights are then updated using backpropagation. This means moving backwards through the network, output to input, and using the loss function to update the weights and biases, with the goal of minimizing the loss function.

## 2.3 Convolutional Neural Networks

A convolutional neural network, or a CNN, is a type of neural network that is mostly used for images and audio [1]. A CNN has the same structure as a neural net with an input layer, a number of hidden layers and an output layer. However the hidden layers are not only fully connected layers, but there are also convolutional layers and pooling layers.

The convolutional layer is the key part in a CNN and does almost all the computation. It consists of the input image, a filter and an output array. The output array is created by taking the dot product between the filter and an area of the input image and then moving the filter one stride until it has covered the entire input image. This process can be seen in Figure 2 where a value in the output array is created by taking the dot product between an area of the input image and the filter. RELU is often used as the activation function in this layer.



**Figure 2** Image showing a convolution in a convolutional neural network

Another type of layers are pooling layers. They down sample the image by applying a filter on an area of the input image, however unlike for convolution layers, the pooling filter does not have any weights and instead either take the average value or the max value of the area. Max pooling is when the max value is chosen and is the most used approach. The purpose of this layer is to down sample the image, which has the side effect of losing information, however the upside is reduced dimensionality and less likely to overfit. As with the convolutional layer RELU is most often used as the activation function.

The output layer is usually a fully connected layer which connects to all nodes in the previous layer. Unlike the previous layers the activation function is usually a softmax instead of RELU.

The ultimate idea behind a CNN is that the first hidden layers will focus on more generic features such as edges and colors, meanwhile later layers will focus on more complex shapes and then finally the last layers will identify the object in the image.

## 2.4 Performance Metrics

### 2.4.1 HTR metrics

To measure how well a HTR model performs, two different metrics are used, character error rate and word error rate.

The character error rate, CER, is defined as:

$$CER = \frac{I + S + D}{n}$$

where I is the number of insertions, S the number of substitutions and D the number of deletions. This gives a percentage of how many of the characters are wrong.

The word error rate, WER, is defined as a percentage of how many of all words are wrong.

### 2.4.2 Classification metrics

Three performance metrics which are used to measure the performance of a classifier are *precision*, *recall* and *F1-score* [23]. The precision indicates the proportion of the positive predicted values which are correct.

$$Precision = \frac{TP}{TP + FP}$$

The recall indicates the proportion of the positive values which was correctly predicted.

$$Recall = \frac{TP}{TP + FN}$$

F1-score is a measure which combines the information from the precision and the recall. This is useful since while looking at them separately, both precision and recall can provide results which does not give a clear picture of how the model performs. A model which predicts only one value as positive and is correct will have a high precision, and a model which predicts everything as positive will have a high recall. To achieve a high F1-score both the precision and the recall needs to be high, thus the F1-score provides a balanced ratio of the two metrics.

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

## 2.5 Handwriting

Handwriting was the way all documents were produced before printing was invented and it is still a method used by many to write text. Since each person can have their own handwriting with their own unique flair and they can often switch between different styles such as cursive or block letters, there are many ways the same text can be written.

## 2.6 Tokenization

Tokenization is a process where each target word is converted to a list of tokens before training. This list needs to be at least two elements longer than the longest word which is being tokenized. This is because each list will start with a special token called START and will end with a token called STOP. For the words which are shorter than the longest word, they will be filled with STOP tokens.

# 3 Related Work

## 3.1 Transkribus

Transkribus is a platform which allows its users to perform text recognition on user provided documents as well as provide other services such as training text recognition models. Transkribus claims to be able to perform text recognition in any language and that it is able to achieve a Character error rate of around 5% [26]. It is not stated which dataset has been used as training and validation datasets. Transkribus allows the users to train their own model by uploading training data which Transkribus then can use to train a model as well as improving their own service. One downside with Transkribus is that the users need to pay for its text recognition feature. Another downside is that when training a model, Transkribus gets access to the training data which could be sensitive material.

## 3.2 Robust Handwriting Recognition with Limited and Noisy Data

While most projects focus on handwritten text that is clearly written and have clearly segmented labels, Pham et al. [17] focuses on text that is limited and noisy. They solve

the problem using a two-stage approach where the first stage is the segmentation of the words, and the second stage is the recognition of the words. The authors explore different segmentation options such as R-FCN [7], R-CNN [10] and YOLO-v3 [21]. The segmentation part only focuses on creating correct bounding boxes and leaves the recognition as a downstream task. Three different word recognition models were tested. The first one was a Word Model that uses an augmented Resnet-18 [11] to predict words from a predefined vocabulary. The second one was a Character Model that uses the same architecture as the first one but uses the CTC loss [12] instead of cross-entropy loss. This enables it to predict sequences of characters, which has the benefit of being able to predict unseen words. The third one learns the embedded latent representation of the images and then decodes this representation into alpha-numeric character, which makes have the same benefit as the Character Model where it is not limited by a vocabulary. It is interesting how the authors separate the segmentation from the recognition, which gives them more freedom in choosing which methods to use. Also worth noting is that the Word Model requires data that is the same language is the images you are predicting on. This makes it difficult to use a word model in this project, since there are no available Swedish datasets that are labeled.

### **3.3 Fully Convolutional Networks for Handwriting Recognition**

Such et al. [24] propose an approach similar to Pham et al. [17] where they first segment the word using R-CNN and then leaves the recognition as a downstream task. The recognition consists of a 3-stage approach using 3 different models. The first stage is a Vocabulary CNN that predicts simple words such as "the" and "her". If the confidence of the prediction is not high enough the image will go forward to the next stage. The second stage is a CNN that predicts the number of symbols in an image. The architecture is similar to the model in the first stage and using the prediction, the authors then resize the image to  $32 \times 128N$  where  $N$  is the number of symbols. The resized image is then sent forward to the third stage. The third stage consists of a dual stream Fully Connected Network that predicts a sequence of characters. This has the advantage of being able to predict any random sequence of characters. The same problem exists in this approach as in [17], where a Vocabulary CNN is used, and this requires a labeled dataset in the language you which to predict on. However, the Vocabulary CNN is not a requirement to get good results. Just using the second and third stage gives results that are competitive with state of the art.



### 3.4 Scan, Attend and Read: End-to-End Handwritten Paragraph Recognition with MDLSTM Attention

Bluche et al. [4] explored the possibilities of creating an attention-based system which is able to do handwritten text recognition on paragraphs, and comparing this with a state of the art non attention-based system when both are trained and tested on the IAM database[14]. The authors developed a system which does not need to segment the input into lines before it transcribes the text. Their model is a an attention-based model and consists of an encoder, which produces feature maps, an attention mechanism, which calculates the weights at every position, and a decoder which uses the feature maps to make predictions.

The encoder consists of a Multidimensional Long-Short-Term Memory Network (MDLSTM) which extracts the features from the image into feature maps. The attention mechanism predicts both the location-based attention and the content-based attention and uses these as a summary of the encoded image. This way the attention weights depends on the context of the entire image. The decoder uses the current image summary and state vector to predict the next character.

The system was first trained to recognize words and lines before moving on to recognize line breaks on text with two lines. When training the system on longer paragraphs the backpropagation needed to be truncated to address the memory issue which occurred. The system was also tested on Arabic text which it was able to read, this shows that it should be able to handle other languages as well such as Swedish.

### 3.5 Boosting the Deep Multidimensional Long-Short-Term Memory Network for Handwritten Recognition Systems

A system which was based on an MDLSTM and an HMM decoding pipeline with linguistic constraints was presented by Castro et al. in [6]. The system takes lines of text from the IAM database as input for its recognition.

The system uses a modified version of an MDLSTM architecture, which usually begins with alternating convolutional and MDLSTM layers. Instead the modified version begins with a convolutional layer and then starts alternating between convolutional and MDLSTM layers, with an extra MDLSTM layer at the end. The two convolutional layers in the beginning aims to increase the sophistication of the feature maps which are fed to the first MDLSTM layer. The authors experiment with the number of hidden layers testing both 6, 8 and 10 layers. For models with more than 6 hidden layers they added an extra downsampling operation after the fourth convolutional layer.

The decoding pipeline consists of an HMM, a vocabulary and a language model. The vocabulary consists of frequent words from the training transcripts, thus it could be problematic to use this model for other languages.

### 3.6 Transformers in Computer Vision

Since *Attention is all you need* [27] was released in 2017 by Vaswani et al, transformers have become the model to use in the world of NLP. In the field of computer vision transformers the same usage have not been seen. In 2021, however, Dosovitskiy et al. showed that on an image classification problem vision transformers could produce results compared to state-of-the-art convolutional networks while at the same time needing less computational resources to train [8].

The vision transformer works by splitting the input image into smaller patches, flattening these patches into 2D vectors and then using these as input to the transformer encoder. However, all the layers in the encoder has a vector size of  $P$ , which means that before the 2D vectors go into the encoder, they first have to be converted to size  $P$  via linear projection. These converted patches are called patch embeddings. To keep the positional information of the original patches a positional embedding is then added to the patch embedding.

The transformer encoder is made up of multilayer perceptrons (MLPs) and multiheaded self-attention layers and will use the patch embeddings with the position embedding as input. The output of the encoder will then pass through an MLP prediction head that will create an output vector of class embeddings containing the probabilities of each class.

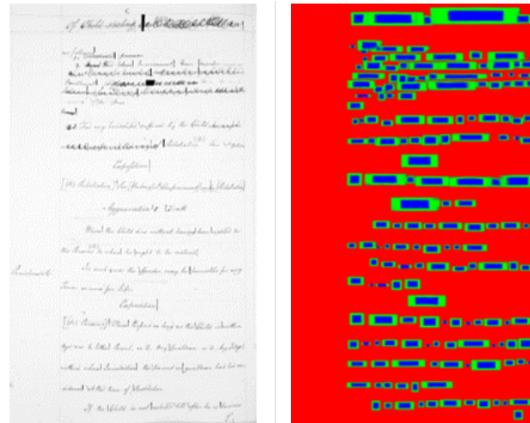
## 4 Methods

### 4.1 Segmentation

The first step of recognizing what it says in a document is to segment all the words into separate images. Since this step was not supposed to be the focus of this project, the goal was to find an easy to use method that gives decent results.

In 2018, Gregory Axler developed a method [3] for word segmentation in documents that achieved state of the art results in segmentation free word spotting tasks. The method works well with historical documents, which is the type of document encoun-

tered in this project. The method consists of three neural networks, where the first one creates a heat map of the document, see following image,

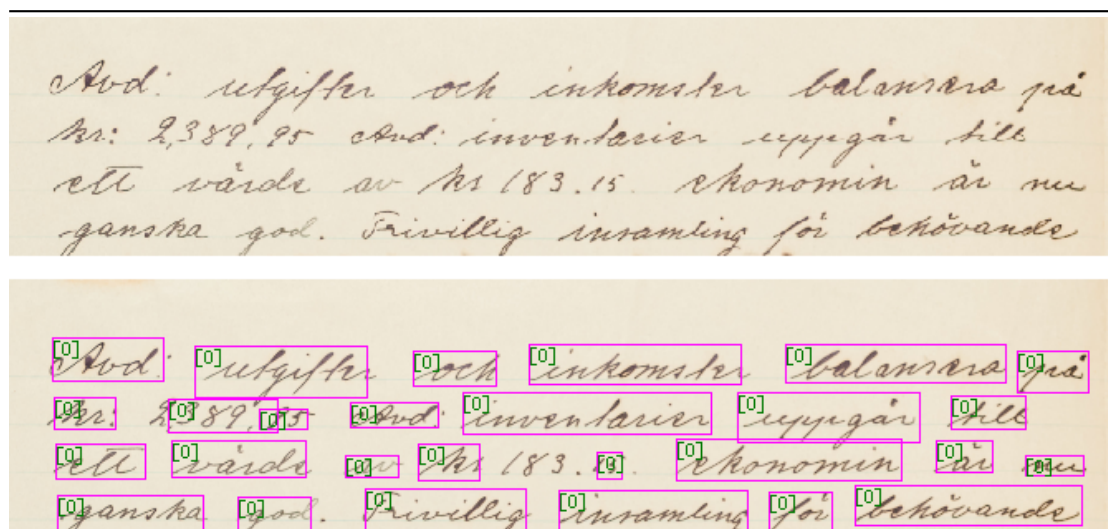


**Figure 3** Example of the heatmap created

where words are likely to be, the second one proposes many possible bounding boxes and the third one filters these boxes and keeps the most likely ones. For a more detailed description see the paper.

The networks are trained using documents as input and the coordinates of the bounding boxes for all the words as targets. This means the method does not care for what the words actually says, just where the bounding box is, which alleviates the need for labelled data sets. The training data used in this project consists of documents from the ICLEF, Botany and IAM datasets.

The result of the method can be seen in Figure 4 and consists of the coordinates for all the proposed bounding boxes.



**Figure 4** Example of document with its bounding boxes

## 4.2 Data Preprocessing

Before the vision transformer can begin its work on the image there are multiple pre-processing steps which are being done to the image first. This is done so the vision transformer can analyze and perform optimally on the input data.

### 4.2.1 Deslanting

Part of the data is written in a cursive handwriting style while other parts are written in a more upright style. To decrease the difference in the data the images of words written in cursive are transformed using DeslantImg [22], which sets these words upright. This will make the text recognition simpler since there will be less diversity in the normalized data.

### 4.2.2 Grayscale

For images which contain colors that are represented in RGB, information about the red, green and blue values of the pixels are stored. This means that computations are made three times, once for each color. By converting the image to grayscale the only information which needs to be stored is the black/white value. This decreases the complexity of

the problem since computations only needs to be done once on the grayscale images. Since images of handwriting consists only of background and words, the color of the images should not carry any relevant information about the words. This means that the images should be able to be converted to grayscale and still contain all the relevant information about the word.

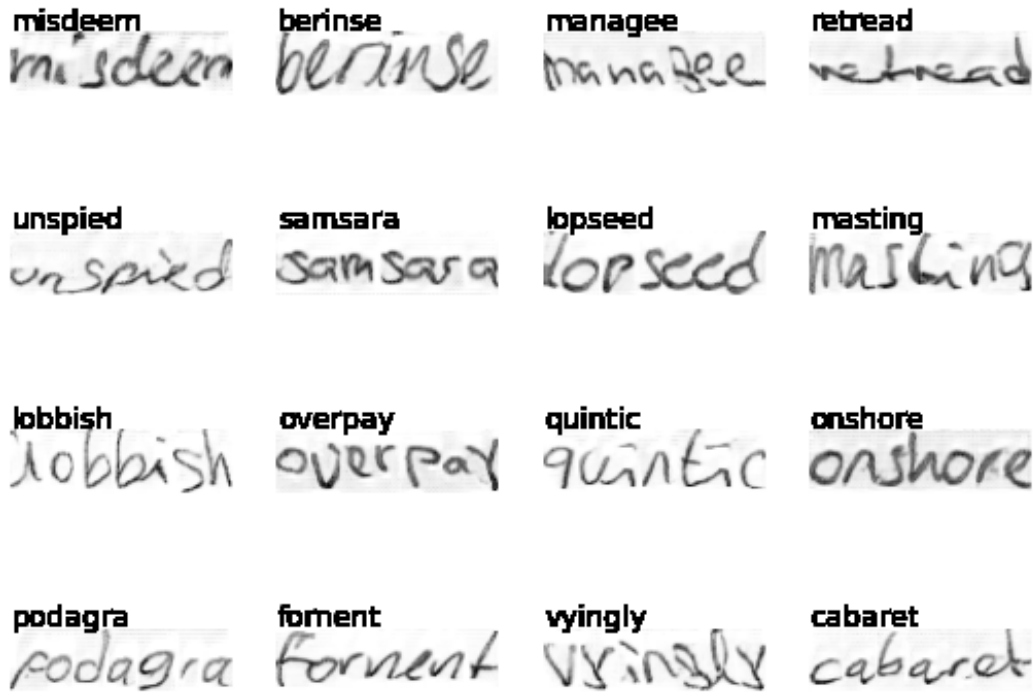
### **4.2.3 Image Size**

The input to the vision transformer needs to be of a certain size, and since the size of the images in the datasets varies, the images needs to be resized to the correct size. One method of resizing the images is by stretching them into the correct size. This means that short words could get stretched wider while long words could be compressed. The other method is to add borders to the images. By scaling the image so it fits either by height or width it is then possible to insert the image onto a white image. The placement of the resized image can be to the left, centered or to the right on the white background. This creates a border either to the right, both sides or to the left of the text. By using all three versions the model learns that the text is not always on the left, center or right side, but that it can be in any of the places.

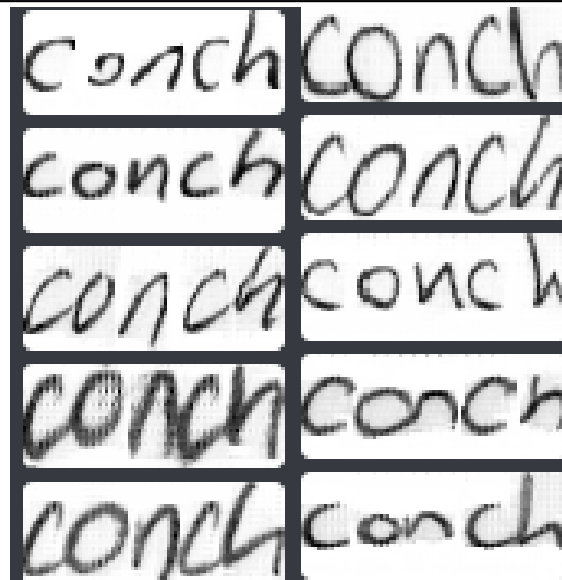
## **4.3 Data Generation**

To generate more data the project ScrabbleGAN was used [9]. ScrabbleGAN is able to generate images of handwritten words with difference in the handwriting. By giving the program a list of words it then generates multiple images of each word with difference in the handwriting. Figure 4.3 shows how one word is generated with 10 different handwriting styles.

In the original work ScrabbleGAN was trained on the IAM dataset [15] to generate images with handwriting similar to the handwriting in the IAM dataset. The IAM dataset was used in this project as well when training ScrabbleGAN. The dictionary which was given to ScrabbleGAN consists of 152 820 English words, thus a total of 1 528 200 images was generated since it generates 10 variants of each word.



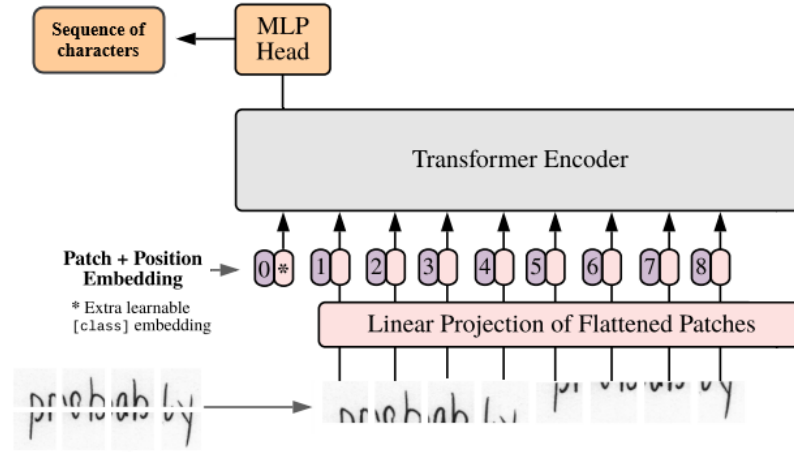
**Figure 5** Examples of words generated by ScrabbleGAN



**Figure 6** The difference between the 10 generated images by ScrabbleGAN

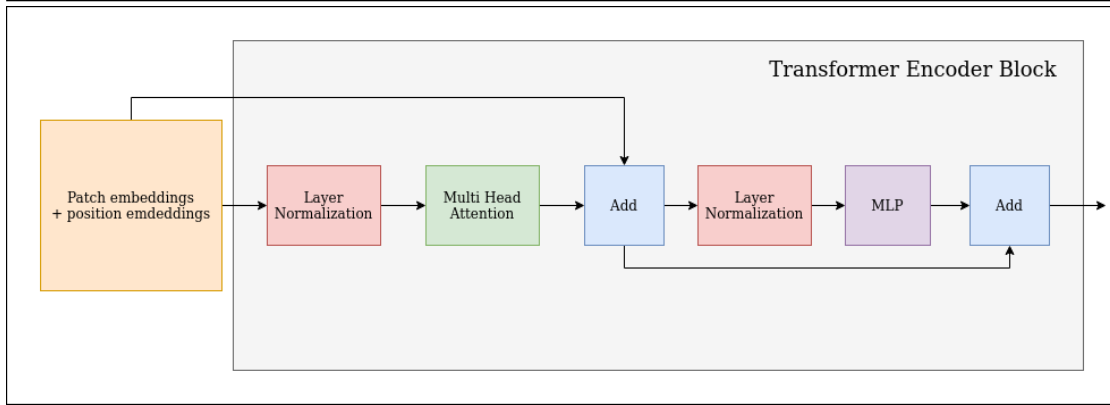
## 4.4 Vision Transformer

The vision transformer used in this project is similar to the one used by Dosovitskiy et al. in *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. The input image of size 32x128 is split into smaller patches of size  $S \times S$ . For example if  $S$  is 8 the number of patches will be 64 and if  $S$  is 16 the number of patches will be 16. These patches are then flattened into 1D vectors and with linear projection converted to size  $P$  to match the layer size in the transformer encoder. These new vectors are called patch embeddings. Then to keep information about which position each patch had in the original image, a learnable positional embedding is added to the patch embeddings. The patch embeddings together with the learnable positional embeddings will be the input to the transformer encoder.



**Figure 7** An overview of a vision transformer

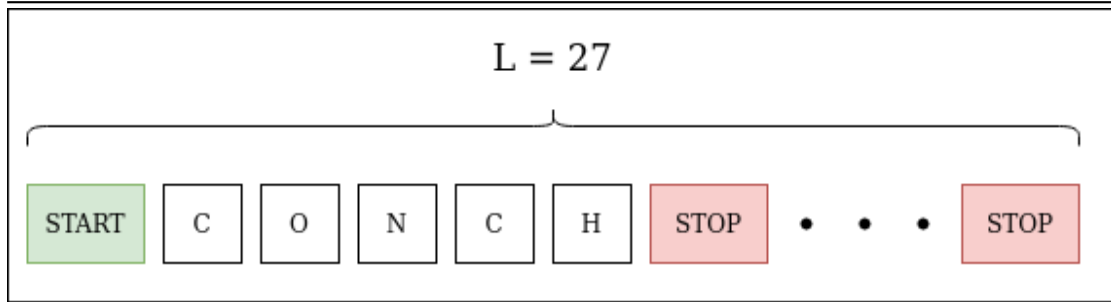
The transformer encoder consists of a stack of  $T$  number of transformer encoder blocks. As can be seen in Figure 8 each block takes the patch and position embeddings as input and uses a combination of layer normalization, multi head attention and a MLP. The multi head attention layer is what calculates the relationship between the embeddings and the number of heads,  $H$ , can be varied. The MLP then acts as a feature extractor and consists of 2 hidden layers with sizes  $P * 2$  and  $P$ . The feature vector is then normalized and this is the output of one transformer encoder block. Then this process is repeated  $T$  times, once for each block.



**Figure 8** A block of the transformer encoder

After the transformer encoder the feature vectors have been created and the last step is now to classify what letters are in the image and in what order. In the original vision transformer made by Dosovitskiy et al. [8], the feature vectors go through a MLP prediction head and then a vector containing the probabilities for each class is created. However since the output in this project is not a single class, but rather a sequence of classes some changes had to be made. The number of classes is 36 and consists of all the characters in the Latin alphabet plus the numbers 0-9. The output from the prediction head should then be a sequence of probability vectors for each position in the word. To do this the target words are encoded before training. Each word is converted to a list of tokens with length 25 (the length is arbitrary, but needs to be longer than the longest word in the dataset), where each letter is one element in the list in the same order as the word. Then if the word is shorter than 25 letters, the remaining elements in the list are filled with a special STOP token. Then a START token is prepended to indicate the start of the word and a final STOP token is appended to indicate the end of the word. This can be seen in Figure 9. With these changes to the target word it is possible to use a MLP prediction head, using RELU as the activation function, that instead outputs a list of probability vectors, where each probability vector gives the probabilities for every letter and number. This list of vectors then pass through a softmax layer to give the final output of a list with 27 elements (25 elements plus the START and STOP token). The list is then decoded by going through the elements and starting the word when the first START token is found and then stopping the word when the first STOP token is found.





**Figure 9** An image showing how the word 'CONCH' is encoded before training

## 5 Experiments

### 5.1 Datasets

The IAM Handwriting Database (IAM) consists of 1539 handwritten documents in English [15]. They have been written by 657 writers thus containing multiple different handwritings. The documents have corresponding ground truths as well as coordinates for each words bounding box. Since there are multiple writers the text in the documents is written in a mix of block letters and cursive, where most of the text is easily readable. The spacing between words and lines is also rather significant compared to other datasets, thus possibly making it easier to separate each word.

The Botany dataset consists of 174 handwritten documents which are written in English [19]. All of the text is written in cursive and the spacing between words and lines vary. The Botany dataset also contains ground truths and coordinates for bounding boxes for the words in the documents.

The ICLEF dataset was used in the ImageCLEF 2016 campaign and was written by Jeremy Bentham (1748-1832)[28]. The dataset consists of 996 handwritten pages from parts of unpublished manuscripts which are written in cursive. It also contains the ground truths and the bounding boxes for each word in the documents.

The test data for this project was provided by the Swedish Labour Movement's Archives and Library. It consists of handwritten documents in Swedish which were written in the late 1800s and during the 1900s. The documents are written by many different authors thus making the handwriting vary between different documents, however they are mainly written in cursive. There are also variations in the amount of degradation of the documents which needs to be taken into account.

## 5.2 Model Training

The models were trained using the Alvis cluster, a SNIC resource which is dedicated to AI and Machine Learning research. The clusters consists of different types of GPUs such as, NVIDIA V100, NVIDIA T4, NVIDIA A100 and NVIDIA A40 [5]. In this project all experiments were done using the NVIDIA V100.

During the time of this project which was 5 months, a total of 9110 GPU hours were used on Alvis. Almost every month was maxed out on possible GPU hours which means there was no opportunity run more tests.

To be able to compare this work to others in a fair way, the IAM dataset with the RWTH Aachen partition was used. This partition splits the dataset into 55 081, 8895 and 25 920 words in a training, validation and test set. These sets are disjoint and there is no writer that is in more than one set. After removing words that have errors in them the final number of images used were 41 233 images plus 150 000 ScrabbleGAN created images for a total of 191 233 images for training, 6225 images for validation and 17 381 images for testing. The validation accuracy was then used to determine which model was the best during training. The time taken until there were no improvements on the validation accuracy during the training was usually between 24-48 hours. The loss function used was the **categorical cross-entropy loss**.

## 5.3 Hyperparameters

The choice of hyperparameters for the vision transformer was done by testing different combinations and then choosing the one with the best result. The parameters that were looked at were: Projection dimension, patch size, number of transformer layers, number of heads, learning rate, weight decay, image height and image width. In a perfect world a grid search would have been done with all values for all parameters, however there was not enough GPU-hours available for this. Instead only **projection dimension, transformer layers, number of heads, patch size** and the **image size** were tweaked while keeping the the learning rate and weight decay for the optimizer as their default values.

The values chosen were the following

**Projection dimension:** 64, 192, 384, 768

**Patch size:** 4, 8, 16

**Transformer layers:** 12, 24

**Number of heads:** 8, 10, 12

**Image width:** 128, 256

**Image height:** 32, 64

A problem encountered when testing was that the graphics card ran out of memory for some models. This is because the models were too big for the graphics cards RAM. This led to us not being able to test some combinations of parameters. In the end the parameters that gave the best result were:

**Projection dimension:** 192

**Patch size:** 8

**Transformer layers:** 24

**Number of heads:** 12

**Image width:** 128

**Image height:** 32

The optimizer used was AdamW and had the default parameters:

**learning rate:** 0.001

**weight decay:** 0.0001

## 6 Handwritten Text Recognition Results

A comparison was done with other attention based models using the IAM dataset with the RWTH Aachen partition, where both the scores with spellchecking and without spellchecking are included

Authors	CER (↓)	WER (↓)
<b>Ours with spellchecking</b>	15.97	23.01
<b>Ours without spellchecking</b>	17.50	32.07
Bluche <i>et al.</i> [4]	12.60	-
Johannes <i>et al.</i> [16]	<b>5.24</b>	-
Kang <i>et al.</i> [13]	5.79	<b>15.91</b>
Suieras <i>et al.</i> [25]	8.80	23.80

**Table 1** Results with a comparison to state-of-the-art methods

An ablation study was done to see the effectiveness of ScrabbleGAN, deslanting and the two resize methods. Table 1 shows the CER and WER for models using stretch as resize method, meanwhile table 2 shows the CER and WER for models using padding

as resize method. Each model was trained three different times on the IAM dataset and the average value for the CER and WER when testing is used.

Models	CER (↓)	WER (↓)
Stretch + Deslant + ScrabbleGAN	<b>15.97</b>	<b>23.01</b>
Stretch + ScrabbleGAN	19.41	27.27
Stretch + Deslant	35.33	39.84
Stretch	42.18	46.67

**Table 2** Ablation study containing the CER and WER for models when stretching the words into the correct size

Models	CER (↓)	WER (↓)
Padding + Deslant + ScrabbleGAN	<b>20.31</b>	<b>27.49</b>
Padding + ScrabbleGAN	20.74	27.90
Padding + Deslant	29.43	35.25
Padding	33.01	40.94

**Table 3** Ablation study containing the CER and WER for models when scaling the word to the correct size and then padding it with white borders

## 7 Complete System

The model used for the system is not the same model which is used when computing the results in section 6. The reason for this is that the complete system is also supposed to be able to predict text which is written in cursive handwriting and in Swedish. The biggest difference in the models is that the model for the system is trained using the datasets Botany and ICLEF as well as the IAM dataset and the ScrabbleGAN generated images. The Botany documents contains 21 981 words, ICLEF contains 75 096, IAM contains 41 233 and ScrabbleGAN consists of 150 000 words. Both Botany and ICLEF contains cursive handwriting more in the style of the documents which the system is supposed to be able to predict on, however neither of them contains text in Swedish. The two different models share the same hyperparameters.

The usage of the system when predicting a documents text is a long process with multiple steps. It takes an image of a handwritten document as input and outputs the words recognized in the document, but internally there is more that happens.

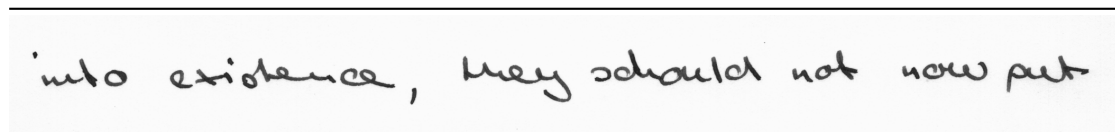
The first step is for the system to take the document and start finding the bounding boxes for the words. This is done by the segmentation tool developed by Gaxler.

After finding the bounding boxes for the words, the coordinates can be used to segment each word so the preprocess steps can be done for each word. The preprocess step resizes, deslants and converts the word to grayscale.

Once the preprocessing is done each word is ready to be sent to the vision transformer to be analyzed. The vision transformer predicts which word it was given and outputs the predicted word.

If desired the predicted word goes through the spellchecker `pyspellchecker`[20] which will compare it to common English words from a dictionary and possibly adjust the word unless it already exists in the dictionary.

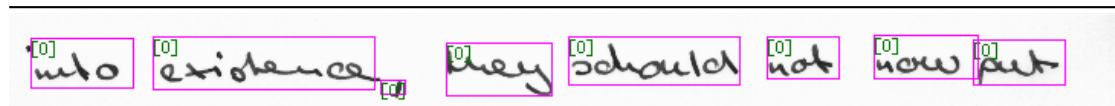
Once all words have been predicted and spellchecked, they are placed on a new document on the original placement of the word. This document with all the spellchecked words displayed is the final output of the pipeline.



---

**Figure 10** One line in a document from the IAM dataset

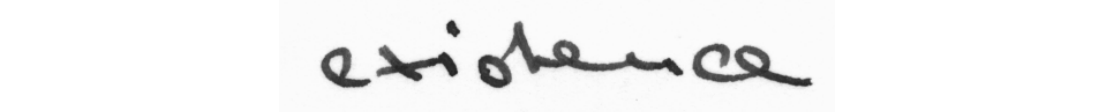
---



---

**Figure 11** The system finds the bounding boxes in the text

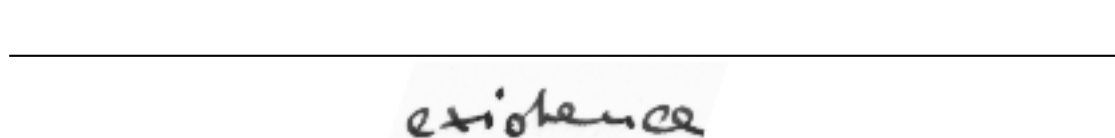
---



---

**Figure 12** Each word in a bounding box is segmented

---



---

**Figure 13** The word is preprocessed

---

exiotence

**Figure 14** The vision transformer makes its prediction of the word

existence

**Figure 15** The prediction is spellchecked and corrected

into existence<sub>1</sub> they could not nowput

**Figure 16** The predicted words are put together again to form the document

*the actual price and expenditure at the said later period as*

the actual<sub>fuuin</sub> and<sub>esperdaien</sub> at the said later fruit is

**Figure 17** An example of the system predicting the text in a line from the ICLEF dataset

*Avd. utgifter och inkomster balansera på  
kr: 2,389.95 Avd. inventarier uppgår till  
ett värde av kr 183.15. ekonomin är nu*

stud uefufe ocn intoomtte balomanss jre  
tii SSOs elud cnronttiin sppoiii tile  
at wand as tas is ehtniilomin SS sec

**Figure 18** An example using text in Swedish from the Swedish Labour Movement Archive dataset

## 8 Discussion

### 8.1 Ablation Study

Comparing the two models which are only using resize methods it can be seen in table 2-3 that the model using padding has a better performance both in CER and WER. The fact that the model using padding performs better makes sense since it keeps the proportions of the word. When stretching the word, it is possible to introduce potential sources of error which are avoided by keeping the original proportions.

When applying the deslanting method to the models they both increase their performance. The padding model decrease the CER by 3.58 percentage units and WER 5.69 percentage units, while the stretch model has a slightly larger decrease in both CER and WER by 6.85 percentage units for CER and 6.83 percentage units for WER.

By adding the ScrabbleGAN dataset to the training the padding model decreases its CER by 12.27 percentage units and its WER by 13.04 percentage units. The stretch model decreases its CER by 22.77 percentage units and its WER by 19.4 percentage units. In this step it can be seen in table 2-3 that the stretch model has passed the padding model in both CER and WER.

Finally, when the models are using both the deslant method and the ScrabbleGAN dataset they both perform at their best. The padding model decreases its CER by 12.7 percentage units and its WER by 13.45 percentage units compared to the model only using padding. The stretch model decreases its CER by 26.21 percentage units and its WER by 23.66 percentage units. Once again it is seen that the stretch model has a greater increase in performance compared to the padding model. Both the padding model and the stretch model performs best when using the deslant method and the ScrabbleGAN dataset in training, though it is the stretch method which has a clear advantage in both CER and WER.

It is interesting that the model which uses stretch as a resize method performs better in the end than the model which uses padding. Stretching the words should have a risk of changing the form of the letters in such a way that would increase the difficulty of recognizing them. However, it might be this that makes stretch the better suited resize method. By training on letters which are both stretched, original and compressed the model gets a larger variety on the training data. This larger variety could be an advantage when the model does its predictions on the test data.

Comparing the results of the model to other work it falls behind the others in both CER and WER as can be seen in table 1. All the other models achieves a lower CER and

only Sueiras et al.[25] has a slightly higher WER. Bluche et al. [4] is using a MDLSTM model while the others are using variations of encoder-decoder models. Kang et al.[13] uses pre-trained weights in the encoder.

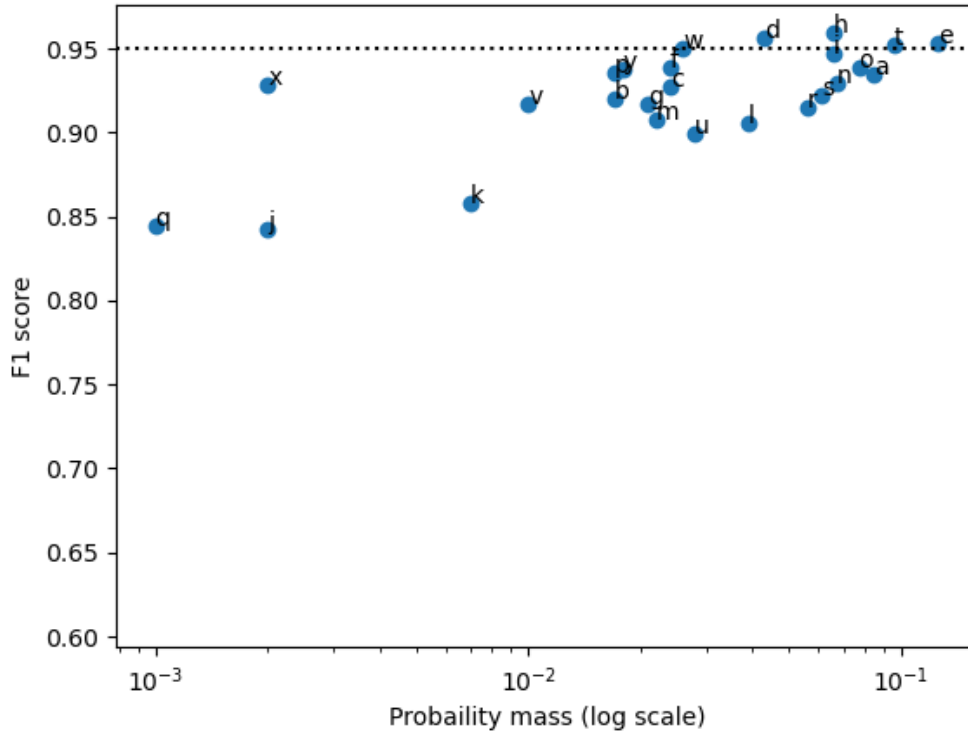
These methods are tried and tested and has previously been shown to perform well on handwritten text recognition, this can not be said for vision transformers. At the beginning of this project no other work using vision transformers for handwritten text recognition was found.

## 8.2 Error analysis for the Vision Transformer

The vision transformer is evaluated at a character level by looking at the recall, precision and the F1 score. This is done by using the images from the test set where the prediction has the same length as the ground truth, which in total were 15 419 images which adds up to 64 599 characters.

Figure 19 shows the probability mass plotted against the F1 score for each character, where the probability mass is the normalized frequency for a character. A dotted line is drawn at a F1 score of 0.95 for visual clarity. What can be seen in the graph is that the letters *q*, *j*, *k* and *v* does not occur very often while also having a lower F1 score. *Z* also has a low probability mass, but the F1 score is too low to be seen in the diagram. This makes sense since if they do not occur very often in the test set, they probably do not occur very often in the training set. The same thing is true the other way around where letters with a high mass probability, such as *e* and *t* also have a high F1 score. The mass probability is however not the only factor in determining the F1 score. Some letters that have the same mass probability, such as *w* and *u*, have widely different F1 scores. This can be explained by the uniqueness of the characters writing and how often they can be confused with other letters. Depending on the style of the writing, *u* might be confused with *o* or *v*, meanwhile *w* is more unique will therefore have a higher F1 score. This can also be seen for *x* where even though it does not appear very often it has a relatively high F1 score which can be attributed to its high uniqueness.





**Figure 19** F1 score shown for each character together with the mass probability on a log scale

Figure 25 in the appendix shows a confusion matrix containing the precision and recall for all characters. The rows show the ground truth characters meanwhile the columns shows the predicted characters. For example the value of  $(q, g)$  is 0.26 and this means in 26 percent of the cases where  $q$  is the ground truth character, the predicted character is  $g$ .

The diagonal shows the recall of each letter and, as seen in Figure 19, further shows how the letters  $q, j, k$  and  $z$  does not score very well.  $z$  is only predicted correctly 28 percent of the time and according to the rest of the values in the  $z$ -row it is often confused with  $e, g, r, s$  and  $t$ , where none of those letters are any similar to  $z$ . This means that the model does not mix up  $z$  with other characters that are alike, rather it cannot recognize  $z$  in the first place. This is different from  $q$  where the recall is 59 percent, which is low, but in 26 percent of the cases the model mixes it up with  $g$ , which is very similar to  $q$ . This instead means that the model can recognize  $q$  but that sometimes handwriting differences

makes it hard to distinguish if it is *q* or *g*.

For characters with high mass probability, the errors are not as big and it is not as obvious which characters are confused with each other. In 4.3 percent of the cases *a* is incorrectly predicted as *o*, which is the highest error of all letters with higher mass probability and this makes sense since they are quite alike.

Figure 24 in the appendix shows the same confusion matrix, but after spellchecking. Here it is obvious that the errors are smaller. Still the biggest errors can be seen for the characters with low mass probability.

### 8.3 Visual Analysis of Images

In addition to the analysis done using the F1 score, recall and precision, there was a visual inspection on a subset of the images from the test set with errors in the prediction. This was done to find reasons for errors that could not be seen in Figures 19, 25 and 24.

Figure 20 shows an image that has an error in the handwriting in the form of a missing letter, which in turn leads to an incorrect prediction. However this was the only example found where this was the case, so it is safe to assume this type of error did not affect much.


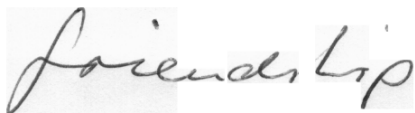
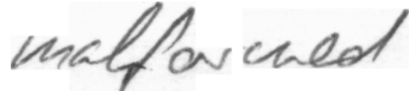




---

**Figure 20** Image with spelling error



---

Bad handwriting is another error that could be found in many examples. The effect of this in the prediction is massive and often leads to predictions that are not even close to the ground truth. This will lower the CER massively since there is often an error in 50 percent of the letters, as can be seen in Figure 21.

	Prediction	Ground Truth
	inefffeceoe	ineffective
	friessee	friendship
	uelforreed	malformed
	aunuatiilly	animatedly

**Figure 21** Images with bad handwriting

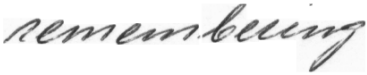


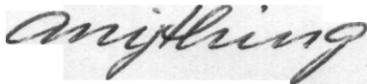
Another form of error that was found was that some images had upper letters in them. This is a problem since there was no consideration in this project to train the model on upper letters. This also leads to predictions that are completely wrong and will lower the CER by a lot, shown in Figure 22.

	Prediction	Ground Truth
	sivinueeeee	somewhere
	mareriu	hopefully

**Figure 22** Images with uppercase letters

Cursive writing is also a big contributor to the low CER as can be seen in Figure 23. The model in this project does not handle cursive writing at all and the predictions are therefore completely wrong. This is a big issue since almost all historical documents are written in cursive, which means the model will not be able to be used for real world issues very well.

---

	Prediction	Ground Truth
	cumuntiuing	remembering
	demmmboiid	disembodied
	conynaa	anyway
	aniliing	anything

---

**Figure 23** Images with cursive writing

---

## 8.4 Analysis of the Complete System

As can be seen in fig 10-16 the system performs quite well on this example from the IAM dataset. Before the spellcheck it predicts "exiotence" which seems reasonable when comparing to the original text. Once the spellcheck is done it has corrected the prediction to "existence". The comma in the original text is predicted as a "1", this is likely since the model is only trained to recognize the letters a-z as well as the numbers 0-9.

Looking at the result of the text from the ICLEF dataset in fig 17, it can be observed that some of the words are predicted correctly while others are incorrect. This could be a cause of the lack of cursive handwriting in the training data. The model was trained using the IAM, ScrabbleGAN, Botany and ICLEF datasets. The combined number of words from IAM and ScrabbleGAN is 191 233 and the combined number of words from Botany and ICLEF is 97 077. This means that a large majority of the training data comes from IAM and ScrabbleGAN which consists of mainly block letters compared to Botany and ICLEF which are written in cursive. Since the model is trained on more IAM and IAM generated data than on the cursive text from the Botany and ICLEF datasets it could lead to a loss in performance while predicting on cursive texts.

When the system predicts on the dataset from the Swedish Labour Movements Archive it is clear that the model cannot handle the handwritten text as seen in the example in

fig 18. One reason could be the lack of cursive handwriting in the training data, another reason could be that the model has not been trained on any Swedish training data. Add to that the fact that the system does not use any spellcheck for Swedish words.

One potential problem with the model is that it uses the same hyperparameters as the model which is used for the results in section 6. By tuning these the performance of the model which is used in the complete system might increase.

The finished system outputs the result by inserting the predicted words on a white canvas at the same coordinates as the bounding boxes of the original words had in the original document. This is an easy way of making sure that the words are placed in the correct order, however there are problems with this solution. The words which are placed on the white canvas vary in size since the font size depends on the width of the bounding box and the predicted word which is to be inserted. A problem which is inherited from the segmentation is that since the bounding boxes sometimes overlap this makes the words overlap each other as well, making it hard to see the covered words. There is also a problem that it is hard to use the information for other purposes when the predicted words are inserted on an image which cannot easily be searched for text.

The reason for this solution is that it works and is easily presentable. Determining the order of the predicted words and inserting them into a document is a challenge of its own. This is made difficult since the bounding boxes vary in size and position. By only using the bounding boxes x- and y-coordinates to determine the order of the predicted words it is difficult to find for example a line break since the y coordinates of the words on a line varies.

## 9 Future Work

One potential problem for the project was the lack of training data, both in English and Swedish. If the model had access to more training data it might perform better. To get access to more data it first needs to be produced, which is a time consuming procedure. The data then needs to be made publicly available and used in the training.

One improvement to the project would be to make the documents searchable by adding the words in order to a searchable document. To do this the order of the words would need to be determined which can be a complicated task. A simpler solution for making a user able to search the documents for words is to put all the predicted words for a document into a text document without regard to the order. This would make it possible to search for a word and finding which documents the word is present in, however since the words are not in order it would not be able to search for phrases.

As a post processing step a spellchecker was used to correct small spelling mistakes, which increased the performance. For future work instead of spellchecking, a language model could be used. This has the advantage of being able to find complex relationships between words and could increase performance by looking at the whole sentence instead of just the isolated word.

## 10 Conclusion

In the end the project answered all the research questions stated.

- Is it possible to create a model used for HTR using a Vision Transformer?

It was possible to use a Vision Transformer as a HTR model. Since it managed to score a CER of 15.97%, meaning it correctly predicts almost 85% of all characters when trained and tested on the IAM dataset, it is safe to assume a Vision Transformer can be used for this task.

- Can the model challenge or outperform already existing models?

It was not possible to create a model that outperforms existing state of the art models. The main reason for this is believed to be that the model used was not pretrained. The model however still performs decently on the IAM dataset, which means there is potential for vision transformers and that more research should be done.

- Can the model be used for real world problems?

For a model to perform well on real world data, especially historical documents, it needs to be able to handle cursive handwriting. However, since there are not many publicly available datasets which contain cursive handwriting and because cursive handwriting often can be ambiguous, it was very difficult for the model to perform well on real world data.

## References

- [1] “What are convolutional neural networks?” <https://www.ibm.com/cloud/learn/convolutional-neural-networks>, accessed: 2022-09-08.

- 
- [2] “What are neural networks?” <https://www.ibm.com/cloud/learn/neural-networks>, accessed: 2022-09-08.
  - [3] G. Axler and L. Wolf, “Toward a dataset-agnostic word segmentation method,” in *2018 25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 2635–2639.
  - [4] T. Bluche, J. Louradour, and R. Messina, “Scan, attend and read: End-to-end handwritten paragraph recognition with mdlstm attention,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, 2017, pp. 1050–1055.
  - [5] C3SE, “Alvis,” 2022, accessed: 2022-07-16. [Online]. Available: <https://www.c3se.chalmers.se/about/Alvis/>
  - [6] D. Castro, B. L. D. Bezerra, and M. Valença, “Boosting the deep multidimensional long-short-term memory network for handwritten recognition systems,” in *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2018, pp. 127–132.
  - [7] J. Dai, Y. Li, K. He, and J. Sun, “R-FCN: Object detection via region-based fully convolutional networks,” 2016.
  - [8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” 2020. [Online]. Available: <https://arxiv.org/abs/2010.11929>
  - [9] S. Fogel, H. Averbuch-Elor, S. Cohen, S. Mazor, and R. Litman, “Scrabblegan: Semi-supervised varying length handwritten text generation,” 2020.
  - [10] R. B. Girshick, “Fast r-cnn,” *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1440–1448, 2015.
  - [11] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, “A novel connectionist system for unconstrained handwriting recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009.
  - [12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.
  - [13] L. Kang, J. I. Toledo, P. Riba, M. Villegas, A. Fornés, and M. Rusiñol, “Convolve, attend and spell: An attention-based sequence-to-sequence model for handwritten word recognition,” in *GCPR*, 2018.

- 
- [14] U.-V. Marti and H. Bunke, “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)*, 1999, pp. 705–708.
- [15] U.-V. Marti and H. Bunke, “The IAM-database: An English Sentence Database for Off-line Handwriting Recognition,” in *International Journal on Document Analysis and Recognition*, vol. 5, 2002, pp. 39 – 46.
- [16] J. Michael, R. Labahn, T. Grüning, and J. Zöllner, “Evaluating sequence-to-sequence models for handwritten text recognition,” 2019. [Online]. Available: <https://arxiv.org/abs/1903.07377>
- [17] H. Pham, A. Setlur, S. Dingliwal, T.-H. Lin, B. Póczos, K. Huang, Z. Li, J. Lim, C. McCormack, and T. Vu, “Robust handwriting recognition with limited and noisy data,” in *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, 2020, pp. 301–306.
- [18] R. Plamondon and S. Srihari, “Online and off-line handwriting recognition: a comprehensive survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63–84, 2000.
- [19] I. Pratikakis, K. Zagoris, B. Gatos, J. Puigcerver, A. H. Toselli, and E. Vidal, “ICFHR2016 Handwritten Keyword Spotting Competition (H-KWS 2016),” in *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016, pp. 613–618.
- [20] PyPi, “pyspellchecker,” 2022, accessed: 2022-11-10. [Online]. Available: <https://pypi.org/project/pyspellchecker/>
- [21] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: <http://arxiv.org/abs/1804.02767>
- [22] H. Scheidl, “2018. DeslantImg. <https://github.com/githubharald/deslantimg>,” 2022. [Online]. Available: <https://github.com/githubharald/DeslantImg>
- [23] M. Sokolova, N. Japkowicz, and S. Szpakowicz, “Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation,” vol. Vol. 4304, 01 2006, pp. 1015–1021.
- [24] F. P. Such, D. Peri, F. Brockler, P. Hutkowski, and R. W. Ptucha, “Fully convolutional networks for handwriting recognition,” *CoRR*, vol. abs/1907.04888, 2019. [Online]. Available: <http://arxiv.org/abs/1907.04888>



- [25] J. Sueiras, V. Ruiz, A. Sanchez, and J. F. Velez, “Offline continuous handwriting recognition using sequence to sequence neural networks,” *Neurocomputing*, vol. 289, pp. 119–128, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231218301371>
- [26] Transkribus, “Frequently asked questions,” 2022, accessed: 2022-06-10. [Online]. Available: <https://readcoop.eu/transkribus/questions/>
- [27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
- [28] M. Villegas, J. Puigcerver, A. H. Toselli, J.-A. Sánchez, and E. Vidal, “Overview of the ImageCLEF 2016 Handwritten Scanned Document Retrieval Task,” in *CLEF*, 2016.



## A Images

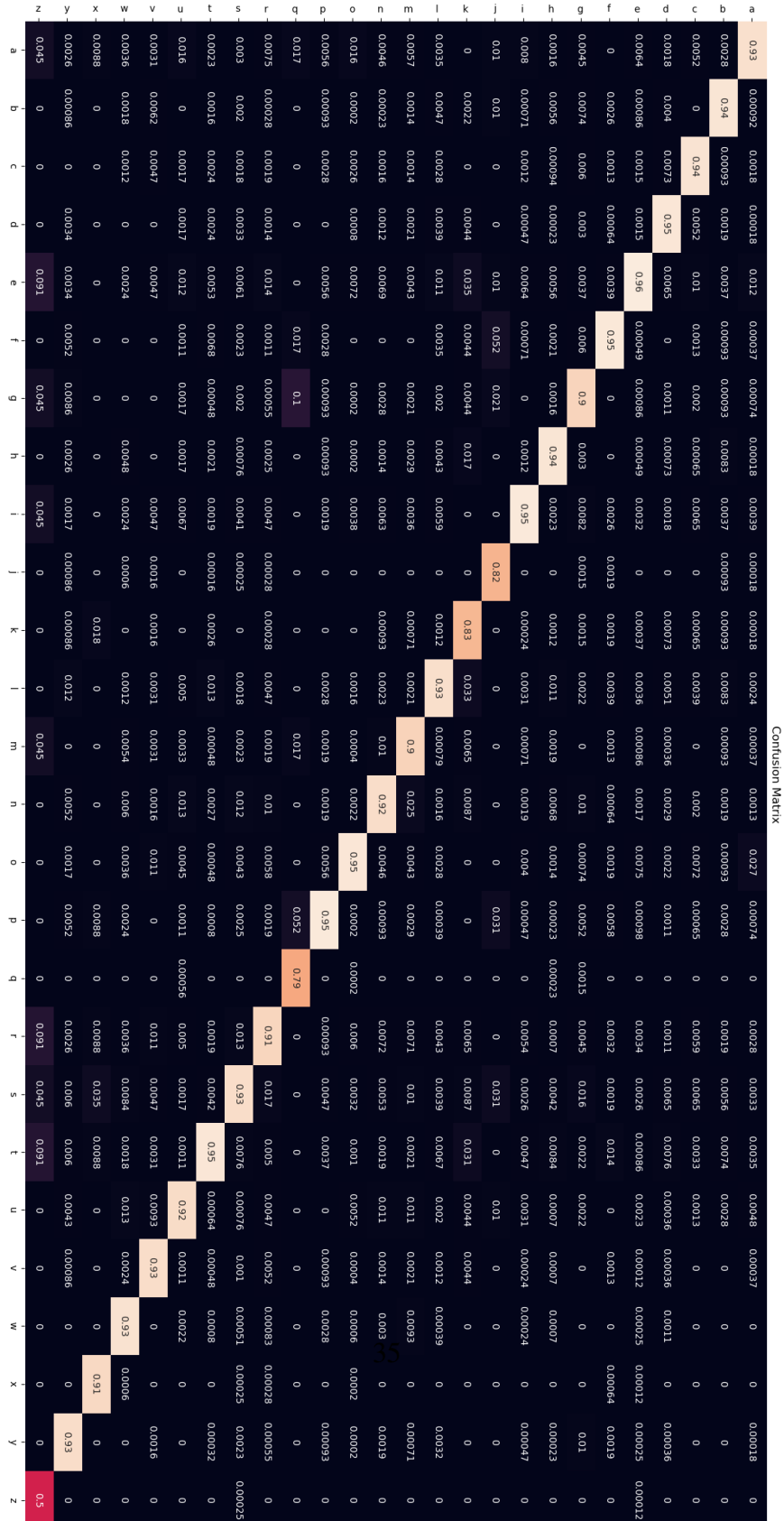


Figure 24 Confusion matrix showing the recall for all characters after spellchecking

Confusion Matrix

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0.9	0.0011	0.0033	0.00074	0.017	0.00056	0.0011	0.00019	0.0039	0.00019	0	0.0032	0.0015	0.0032	0.043	0.00074	0	0.0035	0.0039	0.0035	0.0089	0.00037	0.00019	0	0.00056	0
b	0.0038	0.93	0.0019	0.0019	0.0038	0.0019	0	0.0095	0.0038	0.00095	0.0019	0.015	0.0078	0.0028	0.0019	0	0.00095	0.0057	0.01	0.0038	0	0	0	0	0	0
c	0.011	0	0.91	0.0095	0.021	0	0.00068	0	0.011	0	0.00068	0.0027	0	0.0041	0.0095	0	0	0.0061	0.0082	0.002	0.0034	0	0.00068	0	0.00068	0
d	0.0022	0.0059	0.011	0.93	0.016	0.00074	0	0.00037	0.0037	0	0.00037	0.0059	0	0.0044	0.0048	0.0015	0	0.00074	0.007	0.0067	0.0011	0.00037	0.0011	0	0	0
e	0.0073	0.00075	0.0039	0.0018	0.94	0.00063	0.00088	0.001	0.0049	0	0.00063	0.007	0.0011	0.0019	0.013	0.00038	0	0.0046	0.0043	0.0011	0.0049	0.00025	0.00013	0	0.00063	0
f	0	0.0059	0	0	0.0026	0.94	0.00066	0.00066	0.0033	0.0026	0.00066	0.0072	0.00066	0	0.00066	0.0066	0	0.0046	0.00066	0.019	0	0	0	0	0.0026	0
g	0.003	0.01	0.003	0	0.0097	0.0089	0.86	0.0045	0.012	0.003	0	0.0015	0	0.022	0.00074	0.016	0.0022	0.003	0.019	0.00074	0.0015	0	0	0	0.016	0
h	0.0024	0.008	0.0016	0.00024	0.0059	0.0024	0.0024	0.92	0.0031	0	0.0033	0.017	0.0021	0.01	0.00071	0	0	0.0012	0.0052	0.0092	0.0031	0	0.00024	0	0	0
i	0.0056	0.00073	0.002	0	0.0081	0.0012	0.00024	0.00098	0.94	0.0024	0	0.0036	0.0015	0.0042	0.0069	0	0	0.0066	0.0049	0.0083	0.0059	0.00049	0.00024	0	0.00049	0
j	0	0.01	0	0	0.01	0.08	0.02	0	0.01	0.7	0	0	0	0	0.04	0	0	0	0.07	0.04	0.01	0.01	0	0	0	0
k	0	0.021	0.0046	0.0023	0.046	0.0046	0	0.037	0.0069	0	0.74	0.048	0.0069	0.0092	0	0.0023	0	0.0023	0.0069	0.044	0.0092	0.0046	0	0	0	0.0046
l	0.0053	0.0097	0.0049	0.0053	0.026	0.0061	0.00081	0.0045	0.0069	0	0.0028	0.9	0	0	0.0028	0.0004	0	0.00081	0.0036	0.012	0.0016	0.00081	0.0004	0	0.0057	0
m	0.0081	0.00074	0.0015	0.0015	0.0059	0	0	0.00074	0.0089	0	0.0015	0	0.86	0.0035	0.0089	0.003	0	0.013	0.0052	0.00074	0.0022	0.0022	0.015	0	0.00415	0
n	0.0076	0.00047	0.0017	0.00095	0.0095	0	0.0026	0.0021	0.019	0	0.00047	0.0014	0.019	0.87	0.0062	0.00095	0	0.011	0.0057	0.0017	0.03	0.0014	0.0031	0	0.0031	0
o	0.0022	0.0002	0.0042	0.0016	0.0087	0	0	0.0004	0.0047	0	0	0.0016	0.0004	0.0024	0.93	0.0004	0	0.0065	0.0071	0.00061	0.0065	0.00081	0.00081	0	0.0004	0
p	0.0039	0.00097	0.0019	0	0.0097	0.011	0.0019	0	0	0	0	0.0029	0.0019	0	0.0048	0.95	0	0.0058	0.0039	0.00097	0	0	0	0	0.00097	0
q	0.016	0	0.016	0	0.016	0	0.26	0	0	0	0	0	0	0	0.082	0.59	0	0	0	0	0.016	0	0	0	0	0
r	0.0079	0.0011	0.0039	0.0014	0.024	0.00084	0.00028	0.0014	0.0098	0	0	0.0042	0.0045	0.02	0.0096	0.0014	0	0.86	0.028	0.0084	0.0076	0.0059	0.00084	0.00028	0.0011	0.00028
s	0.0041	0.0026	0.0039	0.0026	0.0088	0.0028	0.0026	0.00077	0.0072	0.00052	0	0.0028	0.0026	0.016	0.011	0.0052	0	0.019	0.89	0.0077	0.0015	0.0013	0.001	0.00026	0.00031	0.00077
t	0.0026	0.0028	0.0023	0.0018	0.0079	0.012	0	0.003	0.0031	0.00016	0.0046	0.024	0.0011	0.0049	0.00066	0.0016	0	0.0028	0.0043	0.92	0.00033	0.00016	0.00049	0	0.00066	0.00016
u	0.023	0	0.0017	0.0011	0.018	0.0011	0.0011	0.0023	0.0051	0	0	0.004	0.0051	0.026	0.0057	0.0011	0.00057	0.0091	0.0023	0.0029	0.89	0.0011	0.0029	0	0.00057	0
v	0.011	0.0079	0.0032	0	0.0079	0	0	0	0.013	0	0	0.0032	0.0016	0.0079	0.016	0	0	0.035	0.0079	0	0.025	0.85	0.0064	0	0.0016	0
w	0.0048	0.0018	0.0018	0	0.0018	0	0	0.0066	0.0012	0	0.0012	0.0006	0.006	0.012	0.0018	0.0012	0	0.0048	0.006	0.0006	0.025	0.92	0	0.0006	0	0.0006
x	0	0	0	0.0096	0	0	0	0	0	0	0.019	0	0.019	0	0.0096	0.0096	0	0.038	0.058	0.0096	0	0.83	0	0	0	0
y	0.0044	0	0	0.0018	0.0053	0.0053	0.015	0.00088	0.0062	0.00088	0	0.023	0.00088	0.0044	0.00088	0.0097	0	0.0097	0.0035	0.0053	0.011	0.0018	0	0.0026	0.89	0
z	0.04	0	0	0	0.2	0	0.16	0	0.04	0	0	0	0	0	0.04	0	0	0.08	0.08	0.08	0	0	0	0	0	0.28

Figure 25 Confusion matrix showing the recall for all characters before spellchecking

## **B Contributions**

Jonathan has been responsible for the segmentation. Martin has been responsible for the data generation using ScrabbleGAN. Both has then split the background work and cooperated to put everything together and creating the vision transformer.