A Pen Is All You Need
Online Handwriting Recognition using Transformers

Matteo Bronkhorst
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
m.bronkhorst-1@student.utwente.nl

ABSTRACT

In 2020, STABILO released the novel OnHW-chars dataset, which contains time series data recordings of characters written with a sensor-enhanced pen. This dataset was accompanied by an extensive exploratory work, presenting baseline accuracies for numerous types of Machine Learning and Deep Learning based classifiers. In 2021, STABILO released a new dataset, containing recordings of written mathematical equations. In this paper, we explore the possibility of applying the recently popularized Transformer architecture to this new dataset. We present a simple but effective adaptation of the Transformer model, where we use two convolutional layers for embedding the input sequence data. With this model, we achieve 92.69% accuracy per predicted individual token. This accuracy successfully highlights the effectiveness of the Transformer architecture in sequence to sequence problems, and encourages further experiments with this model on the OnHW datasets.

Keywords

OnHW, ONHWR, Online Handwriting Recognition, Attention, Transformer, Gated Recurrent Unit, CNN, LSTM, time-series data, sensor-based pen

1. INTRODUCTION

1.1 Background

Written text is found almost everywhere nowadays. Most people can read and write[1], and the recent advancements in Artificial Intelligence have allowed for the development of computer systems that can similarly recognize and produce written text. A well-known example of text recognition can be found in the form of Optical Character Recognition (OCR). The aim of OCR is to convert visual representations of written text into machine-encoded characters. OCR has proven to be an effective approach to digitizing both handwritten and printed texts. One crucial assumption that the functioning of OCR relies on, is the availability of a visual representation of the written text. This means that any time one wishes to digitize written text, some kind of picture must be produced, for example by taking a photograph of the writing.

35th Twente Student Conference on IT July 2nd, 2021, Enschede, The Netherlands.

Types of handwritten texts that are suitable for use in Optical Character Recognition include: postal addresses, forms, forensic evidence, and many more. OCR falls into the category of recognition often referred to as offline handwriting recognition (OFHWR). Offline handwriting recognition stands in contrast to online handwriting recognition (ONHWR), where ONHWR is concerned with recognizing handwritten text as it is being written. In part due to the large body of research in the field of computer vision, OFHWR has seen more extensive research than ONHWR. However, with the recent rise in popularity of personal smart devices, such as smartphones, tablets, and their accessories, the interest in pen-based input technologies has increased.

1.1.1 Pen-based Handwriting Recognition

This has prompted the development of several pen-based technologies, with many of them focused on the combination between a stylus and a secondary device, such as a smart phone or a tablet with touch screen. A novel approach has recently been presented by STABILO, in the form of their STABILO DigiPen[13]. This device is a ballpoint pen enhanced with several sensors, including accelerometers and a gyroscope. The point behind the device is to remove the need for external devices and accessories, such as special writing surfaces. The device is designed to be able to write on normal paper and it records sensor data at a frequency of 100 Hz.

OnHW-chars.

In 2020, the OnHW-chars Dataset was released, as mentioned by Ott et al. in [13]. The dataset contains sensor data recorded with the STABILO DigiPen for 31,275 handwritten characters, 52 classes (26 upper and 26 lower case characters), written by 119 subjects. Ott et al. [13] explored a number of ways that this data can be analyzed, and a fair amount of machine learning models that can be trained on the dataset. The paper focused on several classification methods, and found Deep Learning based methods to be most effective. These models included CNN, LSTM, and BiLSTM networks. These models achieved good baseline performances, with a best result of almost 90% accuracy for a CNN-based classifier for upper case characters. STABILO released the data set mentioned previously as part of the UbiComp 2020 Time Series Classification Challenge. STABILO is currently in the process of releasing a new set of data[16]. This data set concerns a set of sensor data labelled with mathematical equations, with a limited alphabet (digits 0-9 and operator symbols such as =,+,-). The data set is being published in two parts, which are now both available. The first part consisted of 7,180 equations from 36 subjects. The second part added 2,297 equations, from 11 writers, totalling 9,477 equations writ-

ten by 47 right-handed subjects. Each equation is present in the data set as a collection of sequential sensor data, labelled with the complete equation, e.g. "123+123=246". There is no separation of the sensor data into segments that belong to specific characters from the equation, and all equations are mathematically incorrect (meaning that the example of "123+123=246" is *not* present), to prevent a mathematical correlation between parts of the equation. Ideally, the recognition system should not be concerned with the mathematical relation between the numbers in the equation. This second data set is presented as part of UbiComp 2021.

1.1.2 Transformers

One model that was not included in the evaluation of the OnHW data set is the recently proposed Transformer architecture[22]. This novel model type has shown promising results in the field of natural language processing[18, 23] and other fields that focus on sequential information[3]. The model has also been shown to be able to learn how to play chess[11]. The Transformer model is designed to deal with sequential input data, but greatly improves on training speeds compared to other types of sequence to sequence models, such as RNN based architectures. This improvement follows from a big practical difference between Transformers and RNNS: Transformers can process the sequential input data in any order, and allow for significantly more parallelization than RNNS typically do[22].

1.2 Research Question

The main question of interest for this research has been as follows: "Can a Transformer model be applied to the field of Online Handwriting Recognition?" To limit the scope of this paper, we have specifically investigated the applicability for the newly published dataset(s) from STABILO[15, 16], which we briefly introduced in Section 1.1.1.

2. METHODOLOGY

To answer the question posed in the introduction, it is important to carry out experiments with the Transformer architecture on a sequence to sequence translation problem. However, without something to compare the results of those experiments to, it is hard to judge how well the Transformer actually performs on this translation problem. The results from [13] are useful in this regard, but not enough, since they only concern individual characters, rather than sequences of characters. To build a more robust connection between their results and the results from our experiments with the Transformer architecture, we've taken the following steps:

Firstly, we've attempted to reproduce part of the results as achieved in [13], to become familiar with the problem at hand, and the type of preprocessing that has been found to work. [13] gives a clear description of the models that have been used, and of their architecture. They have trained a total of four Deep Learning models: a CNN, an LSTM, a BiLSTM and a combination of a CNN layer with an LSTM layer. In our work, we have chosen to reproduce the results for the best performing and worst performing of these four: the CNN and the LSTM respectively. Convolutional Neural Networks are often used in image processing, because the ability of a CNN to identify simple features in data and aggregate them to more complex parts is very powerful. Long Short-Term Memory is a type of recurrent neural network[12]. LSTMs came about in an attempt to solve the vanishing and exploding gradient problems[14] that plagued many RNNs. Because LSTMs are much better at dealing with these two issues than earlier RNNs have been, LSTMs are very suitable for tackling tasks that involve long range dependencies in sequential data.

Secondly, we've investigated the effectiveness of a sequence to sequence architecture as described in [9]. This involves an encoder-decoder architecture based around Gated Recurrent Units. GRUs are fairly similar to LSTMs, and perform just as well if not better on smaller datasets[5]. This means it should be a good match for the datasets we choose to work with, as a set of around 9000 labeled items is not typically considered large.

Thirdly, we've built a Transformer model, and measured its accuracy. The Transformer is also an encoder-decoder architecture, and has been shown to be very effective in numerous sequence to sequence problems. Transformers come in different shapes and sizes, but in our experiments we've stuck to a reasonably simple architecture. Even though the model was not exorbitantly large, implementing the Transformer required some more effort than expected, which we will further discuss in Section 2.4.

2.1 Datasets and preprocessing

Throughout this research, two datasets have been used for training and evaluation. These datasets both originate from STABILO, who have recently started a new endeavour in the form of STABILO DigiVision[17]. The first dataset was released in 2020, as part of Ubicomp 2020. The second dataset is being released in a similar fashion, as part of Ubicomp 2021.[6]

Both datasets contain sequential data which has been gathered under similar conditions. The first set, as described by Ott et al. is gathered from 119 right-handed persons, who were asked to write the English alphabet six times in lowercase as well as uppercase characters. This resulted in a total of 15,650 lowercase characters, and 15,625 uppercase characters. In total, this adds up to 31,275 handwritten characters. Each character is represented in the dataset by a sequence of features containing 13 values per feature. These sequences are recordings of the sensor values from the DigiPen during writing of the mentioned characters, sampled at a frequency of 100 Hz. Not all sensor have the same measurable range of values, and in Table 1 we list the minimum and maximum values of each sensor. The two accelerometers, the gyroscope and the magnetometer all yield values along three axes. Each triaxial sensor has the same bounds for all three of its axes.

Table 1. Measurement boundaries for the sensors found in the DigiPen.

Sensor	Max value	Min value
Accelerometer front	32768	-32768
Accelerometer back	8192	-8192
Gyroscope	32768	-32768
Magnetometer	8192	-8192
Force sensor	4096	0

2.1.1 OnHW-chars dataset

The dataset from 2020, also known as the OnHW-chars dataset[13, 15], is a collection of six subsets. Each subset contains one of three types of labeled sensor data: upper case characters, lower case characters, or a combination of upper and lower case characters. Another distinction is made between subsets: each subset of data is arranged in a manner suitable for either Writer Dependent (WD) or a Writer Independent (WI) classification tasks. With WI classification, training and validation data are not allowed to overlap in writers. This means that if one character of

a writer occurs in the training data, all characters written by that person are in the training data, and not a single character written by that person is present in the validation data. And vice versa: if a character from a specific writer occurs in the validation data, then none of their characters will occur in the training data. In contrast to WI classification, WD classifications relies on the assumption that the characters of one writer are proportionally spread over both the training and validation set. This division between writers is made based on the idea that different persons have different styles of writing and therefore may produce specific patterns that will only occur in their writing. Even before obtaining any results, it can be expected that Writer Dependent classification will yield better results, because any model will have had the opportunity to become familiar with the patterns of every writer in the validation set. Conversely, the Writer Independent task is expected to yield somewhat lesser results.

An aspect that is mentioned in the readme.txt file that accompanies the OnHW-chars dataset is that each subset is arranged as five tuples for five-fold cross validation. Each tuple consists of four items: training data, training labels, test data, and test labels. As described by the readme, each tuple consists of 100% of the data from that subset, and all tuples contain a different split of training and test data.

2.1.2 OnHW-equations dataset

The newer dataset, which has recently been published as part of UbiComp 2021, contains sensor data recordings labeled with complete equations. We don't exactly know the name that STABILO will decide to give this dataset, we think it would be sensible to name it the 'OnHW-equations dataset', but we will refer to it with 'the dataset from 2021' or something to the extent for the remainder of this paper. The challenge set by STABILO for teams that partake in the competition is to recognize single equations of unknown writers. This would be considered Writer Independent recognition. However, in this paper we focus on the Writer Dependent case. We choose to do so for the sake of simplicity. For the OnHW-chars dataset, the WD tasks have been shown by Ott et al. to give better results. We reasoned that this means that any significant results for our experiments would be visible more quickly under WD conditions, which speeds up the process of debugging and tuning our models.

The dataset is made up of 47 subfolders, one subfolder for each person. Each subfolder contains one folder per recording of that person. Each recording folder contains two CSV files, one with one long timestamped sequence of sensor data, and one with all equation labels and the timestamp intervals that indicate the part of the sensor data sequence that corresponds to that equation. The vocabulary (or alphabet) of the labels consists of digits (0,1,...,9) and mathematical operators (=,+,-,*,:). All writers were right-handed, and the dataset contains a total of 9,477 equations. The dataset is also provided with a Python script which makes it easier to split all recordings into separate labeled sequences. This hierarchical structure of the data is much more versatile than the way in which the OnHW-chars dataset was stored, since it allows for alternative partitioning of data into training and validation sets than either fully Writer Dependent or fully Writer Independent. For example, one could arrange the data in such a way that training and validation data are 50% Writer Dependent, and 50% Writer Independent.

One peculiar aspect of this dataset is that all equations are mathematically incorrect. This is done to prevent abuse of mathematical inference for label prediction. One may argue that this in itself is a hint: a model might learn that if it is predicting a label and has two potential labels, "123+123=246" and "123+123=245", that the latter is more likely to be the ground truth, since the former is mathematically correct and can therefore not be the ground truth. Even though this is theoretically possible, the amount of conceivable equations that are mathematically incorrect definitely far outnumber the amount of conceivable equations that are mathematically correct. Still, this characteristic of the data might be a cause for bias in our models.

2.2 Preprocessing

Before training our models, we've applied a number of steps to transform the sensor data features into a representation better suited for training of Deep Learning models. The preprocessing for both datasets is similar, but differs at some important points. We describe both cases.

The steps applied to the OnHW-chars data is independent of the specific subset that is being used for training. To start off, we make sure that there are no empty sequences. We've encountered some errors that were caused by sensor data sequences of length 0, which was somewhat unexpected. The fix is simple: removing all sequences of length 0 from the set. Subsequently, we resample each sequence to exactly 64 time steps. Lastly, we normalize each data feature. This means that we multiply and translate all values of each feature in such a way that the range of each sensor is compressed into an interval of [-1, 1]. We can do this by multiplying each feature with a vector that maps the ranges in Table 1 to the interval [-1, 1]. This is straightforward for all except one sensor: the force sensor. This sensor has an asymmetrical domain range, meaning that we have to divide each sensor value by 2048 (4096 / 2 = 2048). This gives a range of [0, 2], after which we can translate to [-1, 1] by subtracting 1.

For the dataset from 2021, we extract the CSV data using the pre-provided Python script. This results in a list, where each element represents one of the 47 writers. Each element is again a list, with all the labeled sequences for that specific person. As mentioned before, we chose to explore the Writer Dependent case, so we ignore the distinction between persons, and collect all 9477 labeled items into one list. Then, each item in that list undergoes three steps. First, each item is inspected for any 'hovering', where the force sensor value remains below 200. This lower bound is sufficient to identify a hovering head and tail of the sequence where nothing is being written. Both the head and the tail are removed. Second, we normalize all sensor data features of each item, the same way we normalize the data for the OnHW-chars subsets. Thirdly, we calculate the average length of all sequences, which turns out to be 665 time steps. We double this number to 1330, and we resample every sequence longer than 1330 time steps to exactly 1330 time steps. We also pad all sequences shorter than 1330 time steps with zeros to match the length of 1330.

2.3 Experimental Setup

All models have been implemented in Python, in the form of Jupyter notebooks. There are a number of machine learning platforms, but we have chosen to work with Tensorflow. We've used two setups, both of which were configured with an installation of Python 3.8.5 or higher, including the Tensorflow package, and support for CUDA. The first setup was a Lenovo Thinkpad P51, which fea-

tures an Intel Core i7-6820HQ Processor paired with 16 GB DDR4, and an NVIDIA Quadro M1200 with 4 GB GDDR5. The exact specifications of this device can be found on Lenovo's website[19]. For the first two models, namely the relatively small CNN and LSTM, this setup worked well. However, the significantly larger encoder-decoder architectures turned out to be a fair bit too demanding for this setup. The main limiting factor was graphical memory. This is why a second setup was put into use. The second setup was a distributed computing platform provided by the University of Twente. This platform is available for members of the University of Twente at `https://crib.utwente.nl/geospatialhub/`. This platform is made up of a large number of computing units, where each unit is an NVIDIA Jetson AGX Xavier. The distributed nature of the platform makes it such that there are virtually no limits to the graphical memory available. This allowed us to train much larger models than possible with the first setup.

2.4 Model implementations and training

All the source code and implementations for models mentioned in this paper can be found at `https://github.com/DrumsnChocolate/A-Pen-Is-All-You-Need`. In total, four types of models have been trained and validated. Two of these have been picked from the set of models that were described and evaluated in [13]: a CNN based model and an LSTM based model. A full description of these two architectures will be given below, including any deviations from the descriptions in [13]. These two models were trained and tested on the data from 2020.

The Convolution Neural Network based model consists of two CNN layers, with each a 40% dropout rate, to avoid overfitting. These layers are followed by a fully connected layer of 100 units, with a fully connected output layer of either 26 or 52 classes. To be specific, the recognition of only uppercase or only lowercase characters required 26 output classes, and the recognition of both upper- and lowercase characters required 52 output classes. The CNN hidden layers in this model were each configured with 64 feature maps, a kernel size of 4, a max pooling size of 2. All hidden layers used Rectified Linear Unit activation.

The Long Short-Term Memory based model is similar to the CNN based model. It consists of two LSTM layers, with each a 40% dropout rate. These layers are followed by a fully connected layer of 100 units, and then a fully connected output layer of 26 or 52 classes, depending on the classification problem. The hidden layers in the model were each configured with 64 units. In contrast to the CNN based model, not all hidden layers in this LSTM based model make use of the ReLU activation. Rather, the hidden LSTM layers were configured with the more traditional hyperbolic tangent activation function. The reason for this is that CUDA offers a training speed up for LSTM layers, with one of the requirements for this speed up being that the layers are configured with the hyperbolic tangent activation function. The remaining fully connected hidden layer with 100 units was configured with ReLU.

For the newer dataset released in 2021, again two models have been trained and tested. These are two encoder-decoder architectures. The first of these combines the use of Gated Recurrent Units with Bahdanau[2] additive attention. The second is a Transformer based model. These two models have been implemented with the help of two tutorials from Tensorflow, where *Neural machine translation with attention* [10] gives a practical implementation of the encoder-decoder based on Gated Recurrent Units as described by Chung et al., and *Transformer model for language understanding* provides an example implementation of the Transformer model designed by Vaswani et al. Below, we will highlight the aspects that are relevant for understanding our implementation of the Transformer, but it should be noted that this is by no means an extensive description of how the Transformer works internally. Vaswani et al. splendidly present all the intricacies and explain a great number of variants.

Our implementation of the Transformer is visualized in Figure 6. It features one encoder, one decoder, an input embedding element and an output embedding layer. The output embedding layer is of the form that is often encountered in text generation and neural machine translation. The layer converts input tokens to dense vectors of a fixed embedding dimension d_{model}. The weights and biases within the output embedding layer are learned through training. When we consider the input embedding element, it is first important to establish if it makes sense to use an embedding layer. The sensor data sequences that are used as input are already in a dense representation, so it is not reasonable to apply the same kind of layer as used for the output embedding. To explain this further: the output labels are sequences which are mapped to sequences of integer tokens before being fed to the output embedding layer. These sequences of integers are considered a sparse representation, because there is a lot of numerical 'space' between two integers. By converting these sequences of integers to sequences of dense vectors, we allow the network to learn a more rich representation of each token, and at the same time make this richer representation very compact in numerical space. For the sensor data inputs, we don't need a denser representation of information, since the sensor data is already in quite a dense representation.

However, after numerous experiments without any form of input embedding, we have come to the conclusion that the sequences of sensor data features do not contain enough information in themselves. This is why we introduce two convolutional layers as a way of extracting richer representation from the input sequences. The addition of these layers allows the Transformer to transform the input sequences into sequences of more complex local patterns. We configure each convolutional layer with 128 feature maps, a kernel size of 10, a stride length of 2, and the Rectified Linear Unit activation function.

2.5 Metrics

We use fairly comparable metrics for each model. For all models, we predict one token at a time, which allows us to apply the cross entropy loss function in all situations. We also measure accuracy per predicted token. The use of these two same metrics has kept comparison between model performances very simple and clear. During training, the loss function is of course most relevant for gradient descent, but ultimately we judge the model by its accuracy per predicted token. In this paper, all accuracies mentioned are of this same type: accuracy per predicted token.

We refrain from using attention plots to describe our results, even though we appreciate their powerful representation of the inner workings of the Transformer. A good example of attention plots that allow for better understanding can be found in the example implementation on Tensorflow of the Transformer model: `https://www.tensorflow.org/text/tutorials/transformer#attention_plots`. The reason these plots are so helpful is because they directly show how strongly words in

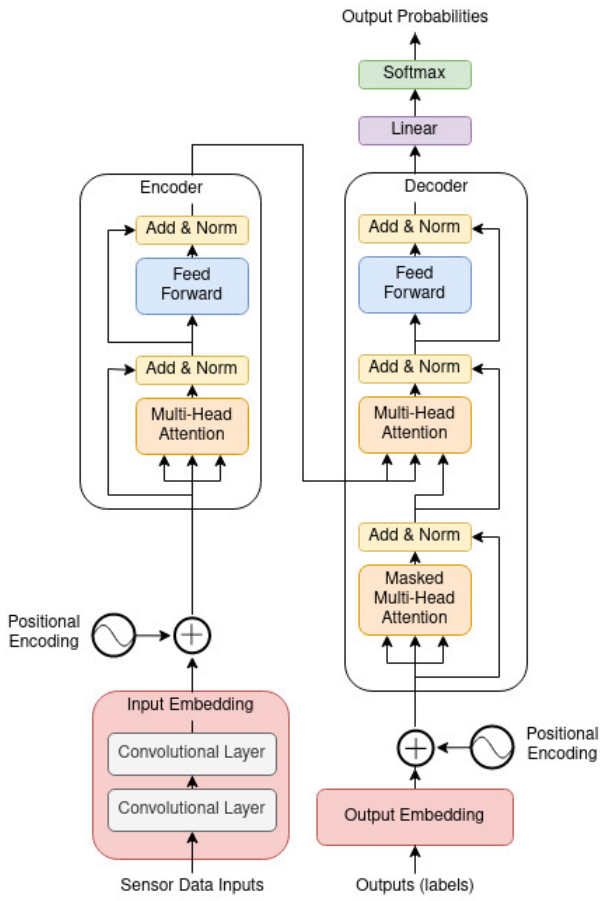

Figure 1. Our adaptation of the Transformer model architecture showcased by Vaswani et al. in [22].

Table 2. Our results for two character recognition models trained on the OnHW-chars dataset.

Method	Lowercase		Uppercase		Combined	
	WD	WI	WD	WI	WD	WI
CNN	82%	73%	87%	80%	69%	61%
LSTM	83%	74%	87%	80%	69%	64%

the source language are related to words in the destination language. In our adaptation of the Transformer, this is not so much the case, since the attention plot would show the relation between the embedded inputs, and the individual tokens of the destination language (the equation labels). The embedded inputs are no longer directly connected to the input sensor data features, meaning that the attention plots no longer directly show how strongly each feature in the sensor input data sequence correlates to individual tokens in the equation labels.

3. RESULTS AND DISCUSSION

3.1 OnHW-chars dataset

The results for the dataset from 2020 are listed in Table 2. Training the convolutional network for 20 epochs took less than 30 minutes per classification task on our first setup, but the LSTM took a while longer to finish. We've not measured the exact duration for the LSTM, but this difference in training time may be attributed to the recurrent nature of the model, which has been known to cause long training times[7]. The accuracies that have been achieved are clearly somewhat lower than Ott et al. achieved. This

	Method	Lowercase		Uppercase		Combined	
		WD	WI	WD	WI	WD	WI
ML-based Features	Random Forest	54.77	43.04	56.29	45.96	42.39	30.44
	Decision Tree	29.36	22.89	29.87	24.32	19.19	14.96
	Logistic Regression	55.36	49.60	58.54	53.26	43.95	39.11
	Linear SVM	61.55	51.07	63.70	54.00	48.77	38.71
	kNN	49.17	29.87	51.32	30.94	38.30	19.61
ML-based Autoencoder	Random Forest	58.02	45.55	63.19	43.73	43.60	43.62
	Decision Tree	30.49	21.73	33.23	19.68	20.32	20.33
	Logistic Regression	56.16	44.93	62.59	43.73	41.66	41.66
	Linear SVM	62.09	51.80	70.61	51.74	46.54	46.56
	kNN	42.43	34.09	57.49	36.68	33.08	33.08
DL-based	CNN	**84.62**	**76.85**	**89.89**	**83.01**	**70.50**	64.01
	LSTM	79.83	73.03	88.68	81.91	67.83	60.29
	CNN+LSTM	82.64	74.25	88.55	82.96	69.42	**64.13**
	BiLSTM	82.43	75.72	89.15	81.09	69.37	63.38

Figure 2. The results from the Online Handwriting Recognition paper published in 2020[13].

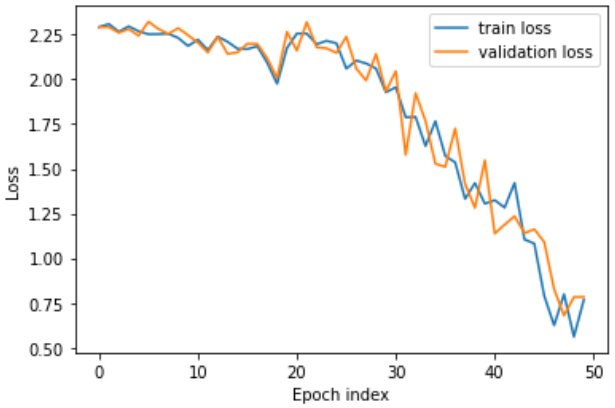

Figure 3. The training and validation loss measured over 50 epochs of training the GRU and Attention based encoder-decoder.

may have a number of different causes that we are unaware of, but there are two things that we can identify which could explain this difference in results. The first of these is that we've trained each model for exactly 20 epochs. Ott et al. doesn't seem to mention the time or amount of epochs that they have trained their models, so it may well be that the performance of their models was similar at 20 epochs. Another possible influence is that Ott et al. mention a correction for bias in the sensor values. It is unclear to us whether or not this correction is already present in the OnHW-chars dataset or not. This also goes for the noise filtering and a number of other preprocessing steps that have been mentioned in their work.

We've encountered some difficulties in replicating the results for the Writer Dependent sets. This seems to be due to an issue within the pre-provided WD files in the OnHW-chars dataset. We come to this conclusion after shuffling the WI subsets to artificially obtain WD subsets. Shuffling the Writer Independent subsets does not guarantee that a writer occurs in both the validation and the training partitions proportionally, but it works well enough for our purposes. This allowed as to sufficiently circumvented the problem, and achieve the WD results listed in Table 2.

3.2 2021 dataset

We've found the GRU and Attention based encoder-decoder to be capable of reaching 82% validation accuracy within 50 epochs. However, the training time required for 50 epochs was around 12 hours, and due to time constraints we have not been able to reproduce this best result. Instead, we provide the loss and accuracy of another training

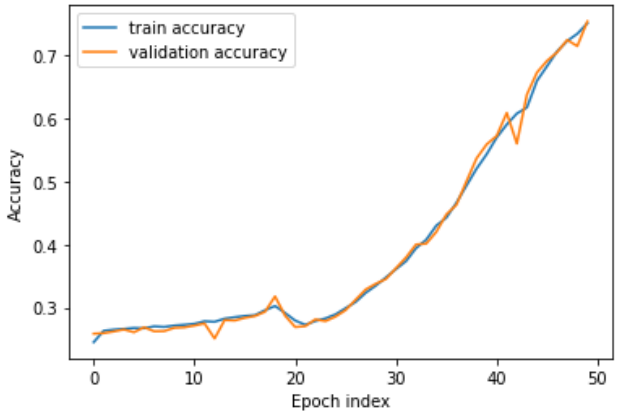

Figure 4. The training and validation accuracy measured over 50 epochs of training the GRU and Attention based encoder-decoder.

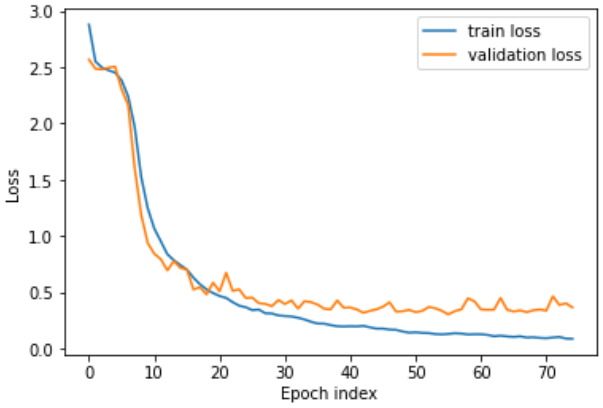

Figure 5. The training and validation loss measured over 75 epochs of training our adaptation of the Transformer

run in Figures 3 and 4. This run reaches 71.04% validation accuracy, which is significantly different from the best performance of 82%, suggesting that the epochs needed for the model to converge are somewhat unstable. However, it should be noted that the curve in Figure 4 has a reasonable slope, and there's no plateau showing yet. This means that the model would likely benefit from training for a higher amount of epochs. We've decided not to do so, due to limited available time, and because these results form a decent enough indicator of the accuracy that the model might achieve if trained more extensively. All in all, these results are not bad at all, but training of this model is rather time consuming, compared to the models we've discussed in Section 3.1.

The Transformer performs very well with the addition of convolutional layers as input embedding, and we've achieved an accuracy of 92.69%. The model was configured with two CNN layers with 128 filters each. This resulted in an embedding dimension of 128. To match this, the output has also been embedded into 128 dimensions per token. These embedded inputs and outputs were then combined with positional encoding, and fed to the encoder and decoder of the Transformer, as described in Section 2.4. The Transformer was configured with 8 attention heads per Multi-Head Attention layer. This model was trained for 75 epochs, and training took 50 minutes, including calculation of validation metrics for each epoch. The values for loss and accuracy for training data and validation data are shown in Figures 5 and 6.

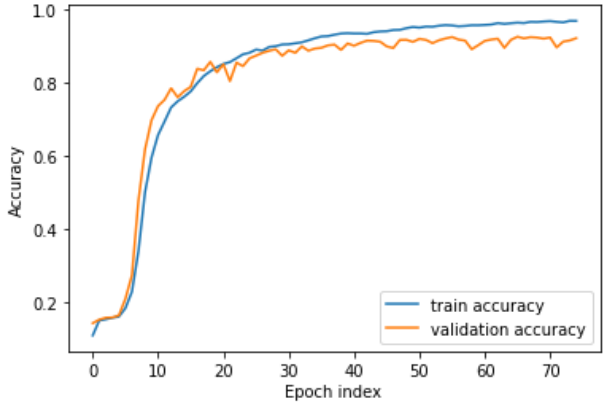

Figure 6. The training and validation accuracy measured over 75 epochs of training our adaptation of the Transformer

4. CONCLUSION

In this paper, we've explored the field of online handwriting recognition, building upon the work of Ott et al. We have shown that their results are reproducible, and we've contributed to the search for solid solutions for the recognition problems that arise from the OnHW datasets. We have outlined, implemented and trained four models, including two encoder-decoder architectures designed for sequence to sequence tasks. With our results, we successfully highlight the effectiveness of the Transformer architecture in sequence to sequence problems, and we provide an adaptation of the Transformer model that embeds input sequences using convolutional layers. This adaptation performs well on the most recently published datset from STABILO, which consists of time series data labeled with mathematical equations. The most notable result in this paper is the 92.69% prediction accuracy that was achieved using our adaptation of the Transformer model, where output labels were predicted per individual character. Our work provides a good basis for further exploration and fine tuning of the Transformer architecture to the datasets from STABILO, and encourages looking more broadly at possible applications for the Transformer to other datasets in the field of online handwriting recognition. All the source code and implementations for models mentioned in this paper have been published at `https://github.com/DrumsnChocolate/A-Pen-Is-All-You-Need`.

5. FUTURE WORK

We've mainly investigated the Writer Dependent case in this paper. It would be interesting to see how well the Transformer model performs in a Writer Independent environment. This should require very minimal work as this can be set up in the preprocessing phase of the experiments; We've already laid the groundwork for an implementation of the Transformer.

Another interesting unexplored avenue lies in the question of how well the Transformer model performs on the OnHW-chars dataset[15]. The Transformer is designed for sequence to sequence problems, and the OnHW-chars recognition problems can be seen as a sequence to sequence problem where each target sequence has length 1.

The embedding section in our adaptation of the Transformer is made up of two convolutional layers, but this

is not necessarily the best approach. It may be the case that there are far better alternatives for embedding the sensor data features that we have not thought of, and it would be valuable to gather more insights in this area. Normally, the embedded sequences are masked during numerous phases in the Transformer, based on the length of the sequence and the amount of padding of the input sequence. We were unable to apply this principle to the features that were extracted with the convolutional embedding layers, as the embedded input was of a different sequence length than the original input sequence. It is not unlikely that this lack of masking based on different input lengths has affected the performance of the Transformer. Further research in this direction could yield ways to re-implement this step.

Besides improving the embedding of the sensor data features, there's a multitude of other ways that one might improve upon the performance achieved in this work. To name a few:

- Artificially expanding the dataset, by duplicating existing samples and slightly shifting/changing the values in the duplicates.
- Applying weight regularization, to try and reduce overfitting.

.

References

[1] *Adult literacy rate*. https://ourworldindata.org/grapher/literacy-rate-adults?tab=chart&time=earliest..latest. Accessed: 2021-05-01.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[3] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. *Is Space-Time Attention All You Need for Video Understanding?* 2021. arXiv: 2102.05095 [cs.CV].

[4] M. Bronkhorst. *DrumsnChocolate/A-Pen-Is-All-You-Need*. 2021. URL: https://github.com/DrumsnChocolate/A-Pen-Is-All-You-Need.

[5] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[6] *Competition Details | STABILO DigiVision*. URL: https://stabilodigital.com/competition-details-2021/.

[7] E. Culurciello. *The fall of RNN / LSTM - Towards Data Science*. Jan. 2019. URL: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0.

[8] *GeospatialHub*. URL: https://crib.utwente.nl/geospatialhub/.

[9] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *CoRR* abs/1508.04025 (2015). arXiv: 1508.04025. URL: http://arxiv.org/abs/1508.04025.

[10] *Neural machine translation with attention*. URL: https://www.tensorflow.org/text/tutorials/nmt_with_attention.

[11] David Noever, Matt Ciolino, and Josh Kalin. *The Chess Transformer: Mastering Play using Generative Language Models*. 2020. arXiv: 2008.04057 [cs.AI].

[12] C Olah. *Understanding LSTM Networks*. Aug. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[13] Felix Ott et al. "The OnHW Dataset: Online Handwriting Recognition from IMU-Enhanced Ballpoint Pens with Machine Learning". In: *Proceedings of the ACM on Interactive Mobile Wearable and Ubiquitous Technologies* 4 (Sept. 2020), pp. 1–20. DOI: 10.1145/3411842.

[14] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. "Understanding the exploding gradient problem". In: *CoRR* abs/1211.5063 (2012). arXiv: 1211.5063. URL: http://arxiv.org/abs/1211.5063.

[15] STABILO. *The OnHW Dataset | STABILO DigiVision*. https://stabilodigital.com/onhw-dataset/. Accessed: 2021-04-30.

[16] STABILO. *Ubicomp 2021 Challenge: Data | STABILO DigiVision*. https://stabilodigital.com/data-2021/. Accessed: 2021-04-30.

[17] *STABILO DigiVision: Digitial pens made in Germany*. URL: https://stabilodigital.com/.

[18] Felix Stahlberg. *Neural Machine Translation: A Review and Survey*. 2020. arXiv: 1912.02047 [cs.CL].

[19] *ThinkPad P51*. URL: https://www.lenovo.com/us/en/laptops/thinkpad/thinkpad-p/ThinkPad-P51/p/22TP2WPWP51.

[20] *Transformer model for language understanding*. URL: https://www.tensorflow.org/text/tutorials/transformer.

[21] *Transformer model for language understanding: Attention plots*. URL: https://www.tensorflow.org/text/tutorials/transformer#attention_plots.

[22] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[23] Shuoheng Yang, Yuxin Wang, and Xiaowen Chu. *A Survey of Deep Learning Techniques for Neural Machine Translation*. 2020. arXiv: 2002.07526 [cs.CL].