Could Font Style Transfer Be All You Need For Historical Document Recognition?

Chenrong Lu
clu582@aucklanduni.ac.nz

Connor Ganley
cgan438@aucklanduni.ac.nz

Michael Smythe
msmy016@aucklanduni.ac.nz

Chong Chuah
cchu742@aucklanduni.ac.nz

Alexander Jefferies
ajef929@aucklanduni.ac.nz

***Abstract*—Historical document recognition of manuscripts with cursive handwriting is a long-standing problem. The main difficulties include lots of visual noise due to the aging of text, faded and skewed text lines. In addition, historical documents contain hard to read fonts that are not readily recognizable by modern optical character recognition methods. This problem is exacerbated by the lack of available labelled training data in the field. In this paper, we propose using generated fake historical data as a solution to the problem of lack of labelled data, and we propose Font Style Transfer as a general training scheme to implicitly solve background noise.**

We conduct some initial experiments with the idea of using Font Style Transfer to turn complex, noise prone historical document text data into a recognizable, regular font.

We compare the result of tesseract OCR before and after the text lines are transferred into a regular font by our method. The OCR accuracy after our method is applied is able to achieve 43.3% character-level recognition accuracy, an 22.9% improvement compared to 20.4% before using our method.

We also investigate the use of the Viterbi algorithm to post-process the OCR results.

***Index Terms*—Historical Document Recognition, Deep Learning, Font Style Transfer, OCR**

I. Introduction

Historical documents are often primary sources of history and are crucially important to understanding the past. Some historical documents also record valuable scientific progress, such as Newton's Philosophiæ Naturalis Principia Mathematica, or Fermat's manuscripts. Yet many people have only heard but not read many important manuscripts first hand, since they are difficult to read and are often preserved in inaccessible archives. Transcriptions and translations can introduce inaccuracies and bias. The ability to properly digitize historical documents autonomously would enable widespread access to primary sources, which may be previously only accessible to experts in historical document restoration. [1]

There are three main kinds of issues in historical document recognition.

Firstly, visual noise due to document degradation. When documents degrade, various types of noise (Fig. 1) occur. The text is often faded or obscured by said noise.

Secondly, difficult to read fonts. Many historical documents are written in obscure fonts that range from medieval Gothic script to extreme cursive handwriting.

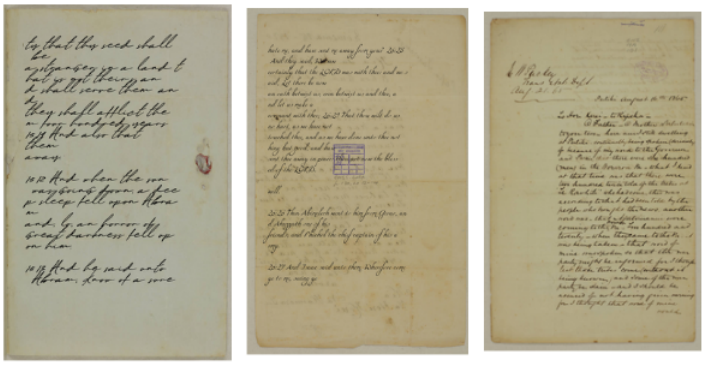

Fig. 1. Examples of historical documents to be recognised.

Thirdly, strange text layouts. Strange text layout issues include font-skew, overlapping words, and complex structural layouts such as the strange geometric formats found in ancient Arabic scripts. [1]

Researchers often regard the third issue as a separate issue in itself. It is the main concern in the field of layout analysis and text-line segmentation. Using manually labelled text lines can also mitigate the third issue.

Therefore the two first issues are the primary concerns of historical document recognition.

Much progress has been made in the field of optical character recognition [32], however many of these methods don't work well on handwritten cursive fonts, and cannot handle well background noise [2].

Recently, a new line of work has begun to emerge, where people have started to leverage results from the rapidly growing field of generative models, on the problem of cursive handwriting recognition.

Specifically, generative models are used to visually transfer a font from cursive into a regular font, and then regular font OCR(optical character recognition) is applied to the transferred text-line.

This idea is beginning to show promise as a viable technique [22] [24]. We elaborate further on in the Related Work section.

For the rest of the paper, we will refer to this new paradigm of doing text recognition as "Font Style Transfer OCR".

It turns out, Font Style Transfer OCR may perfectly solve the two main issues of historical document recognition mentioned above.

Since the learning targets for generative models are clean, regular fonts, the learning task forces the model to learn to deal with the noise implicitly. In essence, Font Style Transfer OCR removes the need for complex pre-processing steps.

In addition, Font Style Transfer OCR allows us to readily leverage mature results in regular font recognition, including accurate, accessible, off the shelf algorithms. Font Style Transfer OCR changes the problem of recognition into solely a problem of regular font recognition, which is a problem generally considered to be solved [2].

Therefore Font Style Transfer OCR may be all we need for historical document recognition.

In this paper, we introduce the first known attempt to apply the idea of Font Style Transfer OCR to the task of historical document recognition.

Our main research question is the following: Can Font Style Transfer help improve normal OCR results on historical documents?
Due to the lack of labelled real data, we simulate this scenario using generated fake historical document data. We evaluate character level accuracy improvements as well as word level accuracy improvements compared with ordinary OCR solutions. We use Tessearct OCR [32] as a main baseline. We also train two different LSTM models from scratch on our dataset, and compare their results as a secondary baseline.

Post processing OCR outputs using language modelling techniques is often used to enhance the accuracy of OCR methods further [1]. Commonly, word level Viterbi algorithm is used [5]. In this paper we examine the use of character level Viterbi algorithm as an alternative post processing technique.

Our main contributions are as follows:

1. We established a novel mock historical document generation scheme. This outputted realistic historical documents to simulate similar noise issues experienced in real historical documents.

2. We trained a single task U-Net architecture on the generated mock data set, transferring a given font into OpenSans Regular. After applying the StyleTransferOCR technique, the accuracy of regular font OCR on the same data set is improved by 22.9%.

3. We trained a few baseline models on the same data set, including ones that are similar to the current state of the art methods in regular font OCR.

4. We tested the model on real historical documents that were trained on the toy data set.
5. We investigate the use of character-level Viterbi algorithm as a post-processing algorithm.

There were other things that we did not end up including in the main method. Section VI serves to acknowledge the other work done by our group members.

II. Related Work

A. Pre-processing techniques

The traditional approach to take when reconstructing historical documents is to first pre-process the documents by removing unwanted features such as noise artifacts from the input image, followed by line segmentation to separate the image into a set of text lines. Conventionally, artifact removal occurs in a single step called "binarization" [7], where suitable pixel activation thresholds work to segment the image into the foreground(text) and the background, removing unwanted noise in the process.

Fig. 2. Examples of artifacts and noise present in historical documents, there are many types of noise prevalent in historical documents, these include but are not limited to :

- Faded Text
- Marginal Noise
- Salt and Pepper Noise
- Low Contrast between document background and text body
- Background Noise e.g. humidity spots, uneven backgrounds

Many methods have been developed to achieve this task. A few of these include Sauvola's method, Nick's method, and Otsu's method [8]. Most of these methods use image thresholding approaches. There are also approaches which use deep neural networks, but these are are computationally expensive [30]. Additionally, traditional computer vision methods such as standard morphological operations can also reduce background noise further. For instance, Messaoud, Abed et al [6] propose a combination of contouring, thinning, and normalization as parts of their pre-processing chain.
These methods work across a large subset of historical documents, but are far from perfect; different approaches work well for some documents but not for others. There is often the need for manual parameter tuning in order to get

them to work well in case-specific situations. [8]

B. OCR (Optical Character Recognition)

OCR is the umbrella term for tasks that involve recognizing digital characters from images [2]. The problem of printed, regular font recognition is now well established and generally considered as solved. However, the same methods that succeeded so well on printed text do not translate directly into cursive fonts, in particular, handwritten fonts [2].

C. Font Style Transfer

Font Style Transfer is the task of trying to transform one font to another while preserving font content. It has applications in new font synthesis, as well as graphic design. The paper by Atarsaikan et al [16] shows that simply applying traditional image style transfer to Font Style Transfer is not enough to produce clear and distinct outputs that retain font structure.
GANs (Generative Adversarial Networks) are often used in this task [12] [19] [17] [18] [20] . Lei Wu et al [20] suggested a very closely related method to ours. Their approach transfers fonts into one another by using multiple U-Nets (U shaped neural network with residual connection) [29] trained on a multitask learning task. Although they seem to achieve visually appealing results, their method is applied on a limited, non-cursive set of Chinese fonts with clear black text and white background.

D. StyleTransferOCR

The task of StyleTransferOCR is to transfer an irregular, cursive font, into a regular font that is easily recognizable by existing OCR solutions.

The motivation for doing this is two fold.

Firstly, this approach leverages the success of existing OCR solutions, which have been trained on enormous amounts of regular font data and fine-tuned to near perfection.

Secondly, this approach leverages results from the rapidly growing field of generative models.
Eman et al (2020) [22] demonstrate that this idea is capable of achieving over 80% accuracy on a synthetic handwritten font recognition data set.

Eman et al's [22] work illustrate that this methodology is plausible as a cursive font recognition method.
Yu et al (2019) [24] evaluated this idea on the declaration of Independence. However, they only tested their method on the 1413 words of the declaration of Independence, and the document they used was in very good condition. Both Eman et al [22] and Yu et al [24] used a Generative adversarial neural network in their experiments.

III. Methodology[1]

Fig. 3. A sample of fonts and blank real historical documents we used to generate fake historical documents.

A. Real Data Acquisition

We began our research by trying to find existing data sets online. Primarily, we sought to restrict our problem domain to handwritten, cursive, 18th Century, English documents. Many labelled data sets available online are not within that criteria. We then tried to acquire historical documents from both the British Library manuscripts online and Auckland Library; totalling approximately 40,000 and 20,000 documents respectively. Unfortunately, these data sets have very few labels and zero segmented text lines.
Manually labelling these text lines is laborious and time-consuming. Therefore, to minimize the reliance on these data sets, we came up with the idea of using generated fake historical data that would provide both labels and text line segments.
This idea is not untested, several OCR techniques for regular fonts are trained on generated data using digital fonts with some cursive font recognition techniques. Our method of training on generated historical data is, to our knowledge, the first application of this paradigm to historical document recognition.

[1]Our Code Repository: https://github.com/IpsumDominum/Is-Font-Style-Transfer-All-You-Need-For-Historical-Document-Recognition-

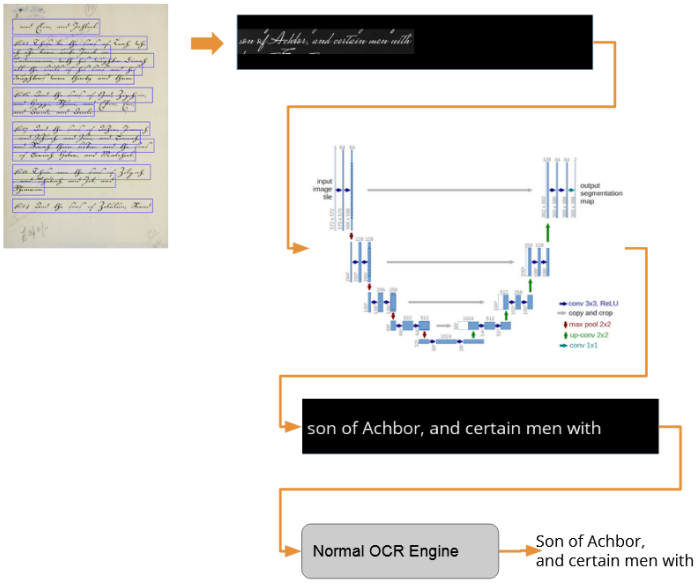

Fig. 4. The entire font style transfer pipeline. Our method: A U-net architecture is used to translate from one padded text line, into an output text image which shows the same text as the text line label, in OpenSans Regular. Then, the transferred image is passed to a normal OCR engine, such as tesseract OCR, which produces the final prediction.

B. Fake Historical Data Generation

The ideal fake historical data for our context should have the following attributes:

1. Fonts should be hard to read and resemble cursive handwriting.

2. Backgrounds should resemble historical documents and capture the old worn out and aged properties.

We used handwritten/historical fonts for generating our data. The method is as follows: First, choose a blank page from a set of blank historical documents, and then overlay lines of text using handwritten/cursive fonts. The text of these documents are lines from the King James Bible, to mimic the 18th century English language. The text lines are wrapped automatically to avoid overstepping the margins of the pages.

In our experiments, we generated 100 documents from each of the 78 fonts, which totals approximately 78000 text lines.

The result of the fake historical data generator can be seen in Fig.1. In fact, only a third of the image in Fig.1 is part of a real historical document, while both the first and second documents are fake. Did we notice when we first looked?

C. Font Style Transfer For Historical Documents

We propose a method to train a general font style transfer model from generated data. The input images are color

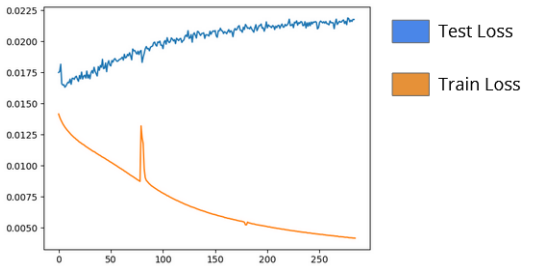

Fig. 5. Loss curve of our training. Due to not keeping a validation set, we did not stop the training and consider regularization techniques. However, the resulting models still generalized on the test data set.

inverted text lines that are padded to shape 80 pixels High by 512 pixels wide. The reason for inverting the text lines is to make high activation equal to text, and low activation to the background.

The ground truth images are images of regular text which contains the same content as the textline labels. The ground truth/transfer target images are all generated using the font OpenSans Regular.

A U-Net architecture is used as the style transfer network. U-Net architectures were originally introduced as a method for

image segmentation [29], but have shown promise in general image to image translation tasks [20]. It was used

by Lei Wu et al [20] in their multitask Chinese character translation experiments. For our experimentation, we used a single U-Net. We trained our model on the generated fake historical document data mentioned above. The U-Net model was trained on 8 Tesla V100s for 40 hours, totalling 285 Epochs.

Our model was trained with common sense hyper-parameters (no hyper-parameter tuning) and non-augmented data.

We used a batch size of 64, and learning rate of 0.01 with the Adam optimizer [31].

Unfortunately, we made the mistake of not keeping a validation set while training. It was too late when we discovered that our model had over-fitted.

Ideally, we would have fine-tuned the hyper-parameters, and applied regularization techniques such as data augmentation and drop out.

Due to time constraint, we did not train our model again.

However, though our model was over-fitted it was still able to extrapolate to our test set, and even on real data. In Section IV we show experiments and evaluations using our

trained model's 74th Epoch checkpoint.

D. Viterbi Algorithm

The output from the character recognition model will contain errors. A natural question to ask is whether there is anything that can be done to reduce the chance of errors? A character recognition model may be seen as having changed the true characters within a document into the observed characters through some statistical process. This underlying statistical process that governs the behaviour of the character recognition may be modelled as a hidden Markov model. The question then is whether implementing probabilistic modelling techniques can improve the accuracy of the character recognition model.

1) Hidden Markov Models: It is first necessary to define what a hidden Markov model is, and how a word may be treated as one. A hidden Markov model is a statistical process where a series of unobserved or "hidden" states influence the observed events. An important characteristic of a Markov model is that when predicting future states, past states do not matter. The current state is the only state which impacts future states. For the rest of this document, a hidden Markov model will be referred to as an HMM. The words output from the character recognition model can be modelled as an HMM by treating the 'true' characters as hidden, and the recognised characters may be treated as the observations. The character recognition model may be treated as a statistical process that changes the hidden states into the observed characters.

2) Definitions for the Viterbi Algorithm: For any HMM, determining the sequence of hidden underlying states that most likely resulted in the observations is required. This is completed through a process called decoding. The most frequently used decoding algorithm is the Viterbi algorithm [3]. The Viterbi algorithm is a dynamic programming algorithm that obtains the maximum a posteriori estimate of the hidden states that resulted in the observed states.
To define the Viterbi algorithm mathematically, the following variables are introduced:
- $\gamma_t(i)$ is the Viterbi path probability at the time step t.
- $\phi_t(i)$ points to the arguments corresponding to the Viterbi path probability at time step t.
- A^* is the probability of the most probable sequence of states.
- x^* is the likeliest path of states.
- π_j is the prior distribution of starting in state j. This will be referred to as the prior distribution for the rest of this report.
- $b_k\left(y_t\right)$ is the likelihood of observation y_t given the state j. This will be referred to as the emission probability for the rest of this report.
- p_{jk} is the transition probability from the previous state x_j to the current state x_k. This will be referred to as the transition probability for the rest of this report.
- N is the number of potential states.
- T is the number of observations.

The Viterbi algorithm may then be defined mathematically in three stages: [26]:
Initialisation:
$\gamma_1(i) = \pi_j b_j(y_1), 1 \leq i \leq N$
$\phi_1(i) = 0, 1 \leq i \leq N$

Recursion:
$\gamma_t(k) = \max_{j=1}^{N} \gamma_t - 1(j) p_{jk} b_k\left(y_t\right), 1 \leq k \leq N\ , 1 \leq t \leq T$
$\gamma_t(k) = \arg\max_{j=1}^{N} \gamma_t - 1(j) p_{jk} b_k\left(y_t\right), 1 \leq k \leq N\ , 1 \leq t \leq T$

Termination:
$A^* = \max_{j=1}^{N} \gamma_t(j)$
$x^* = \arg\max_{j=1}^{N} \gamma_t(j)$

The initialisation stage initialises the values of $\gamma_1(i)$ and $\phi_1(i)$ at $t = 1$. The recursion stage iterates through the potential states to find the probability and states of the most probable sequence of states. The termination stage returns the probability and the states of the most probable sequence of states.
In the context of the character recognition problem, the variables may be defined as:
- $\gamma_t(i)$ is the probability of a sequence of letters at the time step t.
- $\phi_t(i)$ is the letters within the sequence at time step t.
- A^* is the probability of the most likely sequence of letters.
- x^* is the likeliest sequence of letters.
- π_j is the likelihood of each letter in state j.
- $b_k\left(y_t\right)$ is the likelihood of the letter y_t given the letter k.
- p_{jk} is the transition probability from the previous letter x_j to the current letter x_k.
- N is the number of potential letters. At current, this is set to twenty-six, however we can always add additional characters.
- T is the number of characters in an input word.

The transition probability, the prior distribution and the emission probability are required inputs into the Viterbi algorithm.
- The transition probability is constructed through finding the frequency that each letter transitions to each other letter within the King James Bible. To clarify, the letter combination ab involves the letter a transitioning to the letter b. These frequencies are then transformed so that $\sum_{k=1}^{26} p_{jk} = 1$, for $j = 1, 2, ..., 26$. As an example, the likelihood that the letter a transitions into the letter b is stored in p_{12}.
- The prior distribution is constructed by counting the frequency of each letter within the King James Bible. These frequencies are then divided by the sum of all letters within the King James Bible so that $\sum_{j=1}^{26} \pi_j$.
- We hoped to obtain the emission probability from the results of the character recognition model. The emission probability would contain the probability that the recognised letter is every other letter in the alphabet. This would mean

that for each letter, $\sum_{k=1}^{26} b_k(y_t) = 1$. That is, entry b_{11} is the probability that the true hidden letter 'A' is identified as an A, entry b_{12} is the probability that the true hidden letter 'A' is identified as a 'B', and so forth.

The output of the Viterbi algorithm is then the likeliest sequence of hidden letters that resulted in the observed recognised characters. The Viterbi algorithm may be seen as changing uncommon letter combinations to more likely combinations. Uncommon letter combinations, however, may still be a part of a legitimate word. This means that simply running the Viterbi algorithm on every input word might incorrectly change these legitimate letter combinations. A more intelligent implementation is to only run the Viterbi algorithm on words suspected to be incorrect. To achieve this, the Viterbi algorithm is only run on words that are not within the English dictionary.

IV. Discussion and Results

A. Testing Data

We tested our model on 1632 segmented text lines from our fake historical data, which is 2% of our training data.

B. Font Style Transfer Results

While our model showed over-fitting on testing scores, evaluating the results visually shows that albeit over-fitting, the model was still able to generalize somewhat on the test data-set. This shows that model is capable of learning a generalized solution to the task. Suitable regularization techniques can potentially improve this.

1) Noise Removal: In Fig.6. One may observe that a multitude of noise exist in the test data, including unwanted branches of adjacent text-lines(Sample 1 and 4) and occluding background noise (sample 2). These noise did not persist in the output text, they are effectively removed. This addresses the first large issue of Historical Document Recognition.

2) Hard To Read Fonts: Our model is capable of transcribing extremely cursive fonts, such as in sample 3, into a recognizable regular font.This addresses the second large issue of Historical Document Recognition.

C. OCR Evaluation

We empirically evaluated the performance of our model by running TesseractOCR [32] on the transferred text line images and evaluated the result by the following metrics:

- Character-level accuracy: Percentage of characters correctly predicted. During comparison, we ignored white spaces.
- Word-level accuracy: Percentage of words correctly predicted.

Fig. 6. A sample of some of the outputs of our model, vs ground truth. The top row is input, the middle is ground truth, and the third row is model output, for Epoch 74 checkpoint and Epoch 285 checkpoint respectively. We evaluate our model on the Epoch 74 checkpoint.

We evaluated the accuracy for each of the samples and then calculated the average. Equation 1 shows how this score is calculated. Where EvaluationType refers to either character-level or word level accuracy.

$$TotalAccuracy = \frac{\sum EvaluationType(sample_i)}{numsamples} \quad (1)$$

We compared our results to the output of Tesseract OCR on the original text lines, prior to the style transfer. This served as the primary baseline.

Note that the theoretical perfect accuracy of our method is limited to the accuracy of the regular font OCR engine that is used. In the evaluation we refer to the theoretical perfect

accuracy as "Tessearct OCR on Regular Font". Specifically, this score is obtained by running Tessearct OCR on the ground truth images in our Test Style Transfer Dataset.

Complimentary to the main baseline we have also included the scores of some other methods that we implemented, all of which were not successful in training on our data set. Since their results were poor, we did not want to include them so to distract the main method descriptions. Their implementations will be elaborated in Section VI.

Results		
Method	character-level Accuracy	Word Level Accuracy
Baseline (Tessearct OCR)	20.4%	17.3%
Patch LSTM	2.33%	0.6%
Attention LSTM	3.38%	0.0%
Style Transfer + Tesseract OCR	43.3%	33.6%
Tesseract OCR on Regular Font	99.6%	93.7%

D. Evaluation On Real Data

While we could not systematically evaluate our model on real historical documents, we ask the question, has the model learned to extrapolate at all, and if so could it learn to recognize fonts in real historical documents?
We observed that while the model in general could not understand full sentences, it did much better on individual words. Please refer to Figs 7 and 8 for a set of visual results. These results were generated from manually taking a crop of some parts of real historical documents, and passing it into the model.

E. Viterbi Evaluation and Results

As mentioned previously, the Viterbi algorithm requires knowledge of an emissions matrix to determine the likeliest sequence of characters. It was hoped that this matrix could be retrieved from the output of the character recognition model, however this proved to be more difficult than anticipated. As a consequence, the developed Viterbi algorithm could not be integrated into the rest of the pipeline.
Despite being unable to connect with the pipeline, results

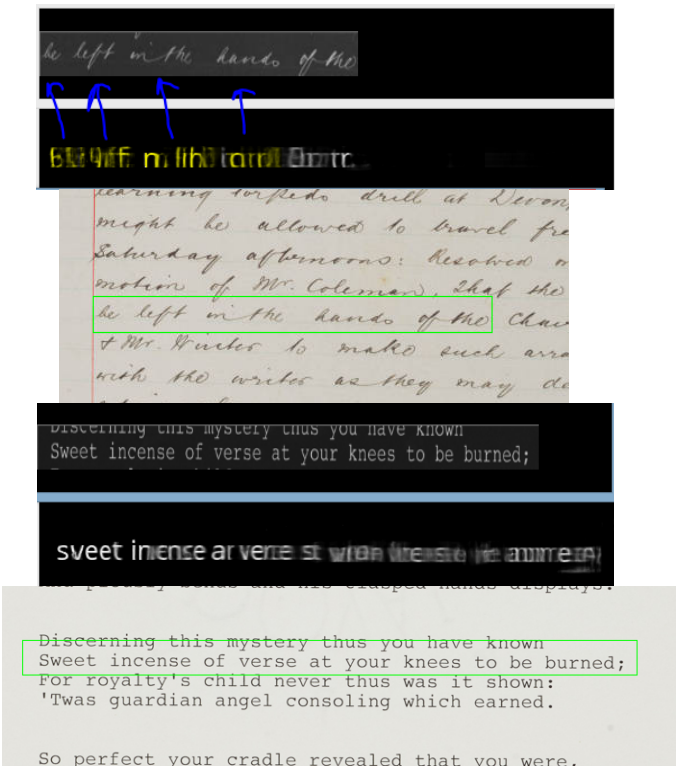

Fig. 7. Some samples of sentence level real historical document outputs of the Style Transfer Algorithm. Top two black lines are input and model output respectively, the screenshot corresponds to where the input text is segmented from.

were obtained from running the Viterbi algorithm in isolation. We achieved this by constructing a known emissions matrix and changing a series of input text based on the corresponding probabilities. The input text consisted of a subset of 24,000 words from the King James Bible. This input text simulated the results of a character recognition model. Again, let us emphasise that these results do not represent the impact of the Viterbi algorithm on the overall pipeline, but on known, well-behaved parameters.
When analyzing the results of the Viterbi algorithm, the following definitions were used:
- A true positive occurs when a character is correctly unchanged.
- A true negative occurs when a character is correctly changed.
- A false positive occurs when a character is incorrectly unchanged.
- A false negative occurs when a character is incorrectly changed.
- The "correct character probability" refers to the probability that an individual character is correct.
- The "correct word probability" refers to the probability that each character within a word is correct.

Fig. 8. Some samples of word level real historical document outputs of the Style Transfer algorithm. Top two black lines are input and model output respectively (Note for sample 1 and 2 the order is reversed), the screenshot corresponds to where the input text is segmented from.

Input Correct Characters	Output Correct Characters	Input Correct Words	Output Correct Words
0.5	0.67	0.11	0.4
0.6	0.73	0.18	0.46
0.7	0.79	0.29	0.54
0.8	0.86	0.44	0.64
0.9	0.93	0.67	0.78

As may be seen in the table above, the initial results indicate that the Viterbi algorithm might benefit the results of OCR. On an input data set where it was known that half of the characters were incorrect, the Viterbi algorithm could theoretically increase this proportion to two thirds. When half of the characters were incorrect, only eleven percent of the words had every character correct. With Viterbi, this increased to forty percent of the words being correct. These increases also occurred in data sets with up to 90% of the words being correct, although the improvement was less substantial.

Input Correct Characters	True Positive Rate	True Negative Rate
0.5	0.88	0.68
0.6	0.92	0.62
0.7	0.95	0.59
0.8	0.97	0.55
0.9	0.99	0.48

As seen in the table above, an input data set with half of the correct characters achieved a true positive rate of 88%. In the data set with 90% correct characters, this rate improves to 99%. This high true positive rate is caused by only running the Viterbi algorithm on words that are spelt incorrectly. Any word spelt correctly in the original data set would not be altered.

On the input data set with half of the correct characters, the Viterbi algorithm has a true negative rate of 68%. When the proportion of correct characters is increased to 90%, this rate decreases to 48%. This makes sense, as there are comparatively less obscure letter combinations in the more accurate input.

As mentioned, these results were obtained on characters that were changed with respect to a known emissions matrix. Consequently, these results should not be viewed as the expected response if incorporated successfully with the rest of the pipeline. Instead, these results demonstrate the theoretical results that could be obtained with knowledge of the true parameters.

V. Limitations

The primary limitation of our method is that for real-world results we are dependant on how well the generated data mimics the real data. On the contrary, we expect the model's ability to generalize to improve as we introduce more fonts and more training data.

Another limitation of our method is that our model does not perform well when the input text line is not properly segmented (if the font is not in the centre of the input). Training on augmented data may alleviate this.

Due to the fully convolutional nature of the U-Net, our model will theoretically work on all input sizes. However, we haven't tested our model on arbitrarily sized inputs. To counter this, we can fragment a longer sentence or run our model multiple times.

Another limitation is that our model is limited to transferring the characters within the English language. We have not tested how well our model performs across other languages.

One possible other concern is the accuracy of the OCR engine applied at the end of the transfer process. However, in practice, if the style transfer process can successfully transfer the hard-to-read fonts into OpenSans Regular, then as shown in the results table, current OCR solutions can achieve $> 99\%$ accuracy. Therefore we argue that this isn't actually a concern.

Several issues limited connecting the Viterbi algorithm to the results of the character recognition:
- The emissions matrix could not be obtained from character recognition.
- Letters could be recognised as either no characters or a concatenation of more than one character.
- The Viterbi algorithm is conservative in changing characters.

VI. Other things we did

A. Manual Data Labelling tool

We wrote an online, collaborative data labelling tool and used it initially to label authentic historical documents.

B. Automatic Text segmentation algorithm

We came up with an algorithm that can automatically detect regions of text. e came up with an algorithm that can automatically detect regions of text. The algorithm works by the following: The image is first inverted then iteratively eroded. The inversion makes sure that the text regions are high pixel activations. Since text regions often have a similar surface area, they tend to disappear at similar rates, compared to that of other types of image patterns, such as blobs of colors or speckle noise. Therefore after the iterative erosion, the average pixel activation values (density) of text like regions would become distinguishable to that of other types of patterns. Therefore, a suitable cutoff threshold can be manually found, such that when looping over the image with a sliding window, only sliding windows with density within a certain range is a text region. Thereby the algorithm segments the image and extracts text regions. This algorithm requires tedious manual tuning of the text threshold, so it wasn't used in our pipeline.

C. Attention LSTM

LSTM is a neural network architecture that works similarly to a recurrent neural network. LSTMs are typically used on sequenced data; they are regarded for their ability to store information in their neurons. The models we implemented follow a methodology similar to the method proposed by Kelvin et al (2016) [27]. The image line segment is first run through a pre-trained Convolution ResNet to extract 10 x 64 sized feature maps, these sequences of feature maps are then fed into a LSTM. Traditional encoders and decoders are not able to handle long sequences of data. Thus, an attention mechanism is implemented. The attention mechanism linearly sums the sequences to allow the encoders to see the sequences. This model is trained for 16 Epochs with a batch size of 64 on 8 Tesla V100 GPUs.

The model did not perform as well as we expected. The training loss function only decreased slightly and quickly plateaued. We believe that this occurred due to the model's lack of power. The model was evaluated using the Mean Square Error loss. The evaluated results are shown in the Experiments and Results section.

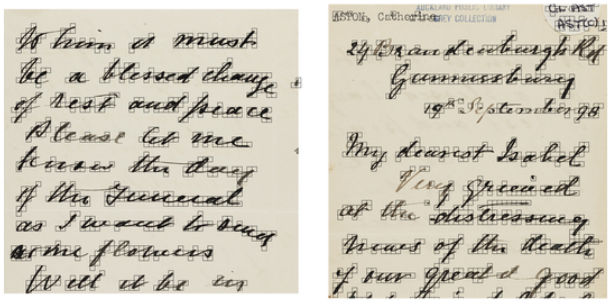

Fig. 9. Sample outputs from our text region segmentation algorithm. Text like regions. Black bounding boxes are applied over text regions.

D. Patch LSTM

This model is designed similarly to the current state of the art method in a way that the neural network reads patches of the image. However, we did not use a CNN and CTC loss to encode image patches and align them as illustrated in Figure 10. Instead, we trained the model with a batch size of 1 and used a varying crop size such that the cropped image corresponds to the output length. The neural network architecture used is an LSTM; the model was trained for 15

Epochs.

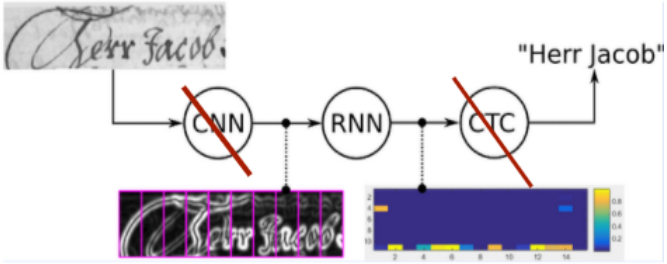

Fig. 10. An illustration of the differences between the SOTA method and our Patch based LSTM.

The model is able to perform well in predicting certain words. However, the training loss plateaued very quickly, which shows that the model is not suitable. The evaluated results are shown in the Experiments and Results section.

E. EasyOCR

We tried to use EasyOCR [25] as another baseline to form comparisons among our models. We created and trained three LSTM models using the EasyOCR open source code [25]. We trained three models. The first two models were left training for about three to six hours each. The first model was trained by taking the generated fake data and splitting them into 4/5 training and 1/5 validation and the other was trained using 9/10 training and 1/10 validation. There were other attempts at trying to train models but they all performed abysmally each giving us scores of 0.000 accuracies with confidence of 0.000. A third model was left overnight to train on the 4/5 testing and 1/5 validation for roughly 8 hours showing absolutely no improvement from the previously trained models. All the models gave equally strange predictions. For example, the letter ”n” would be predicted to be ”tttzzyyzzzzzzzzzzzzzzzzz” and the word ”long” would be predicted to be ”55ccgcgcgcgcgcgcgcgcgcg”. From these observations we suspect that the problem is that the benchmark is training on line segments instead of character segments, hence the strange recursive z’s and cg’s. A possible solution would probably be to generate not only line segments but also word and character segments. In this way, the model learns cursive handwriting by characters instead of lines. However, when applying the model to the real-world, if we were to use real documents we would need a text segmentation algorithm that can both segment cursive handwritten texts by lines and segment documents by words and characters, which we have found to be outside of the initial scope for our style transfer OCR.

VII. Conclusion

We propose a novel method of doing historical document recognition using FontStyleOCR.
We propose a method of generating fake historical document data for training the FontStyleOCR pipeline.
We trained a U-Net architecture on the Font Style Transfer task on our generated fake historical document data.
Using our method, we are able to improve regular font OCR accuracy on our generated data set by 22.9%.
We conducted preliminary experiments and tested our FontStyleOCR model on real historical document data and show that it generalizes somewhat.

Our proposed method showed promise in resolving the two main historical document recognition problems that we set out to address. Our model was able to recognize characters without any binarization and still managed to remove noise. An example of noise removal can be seen in the second example in Fig.6. Here the word ”night” is partly occluded by background noise, but in the model’s normal font output, the noise was successfully removed. We were surprised to see that our model can understand extremely cursive fonts, such as the third example in Fig.6.

Although our model did not achieve notably good accuracy, nor was it able to perform well on real historical data, we argue that our results have still validated the principle of using fake generated data to train a model that works on real historical documents. As shown in figure 7 and 8, our model produced results that only slightly degraded from our test samples. This suggests that the model’s performance on real historical documents is indeed correlated with its performance on the fake data set

The implementation of the Viterbi algorithm was not successful in the pipeline. As a probabilistic process, the emissions matrix was required to successfully maximise the sequence of letters. As the emissions matrix was unable to be returned from the character recognition models, the Viterbi algorithm was not able to run on our recognised characters. In addition, the output from the character recognition often resulted in an individual character either not being recognised, or being recognised as multiple characters. We can still run The Viterbi algorithm in these cases, but assuming that each letter is only one letter limits the potential in correcting character recognition.
When we input a known emissions matrix, theoretical results for the Viterbi algorithm were obtained. These outputs suggested that the Viterbi algorithm with ideal parameters:
- Is capable of improving the accuracy on both a per-character and per-word basis. This increase in accuracy is more pronounced on initially less accurate predictions.
- Infrequently changes correct characters, and is capable of correcting the other half of the incorrect words. The more accurate the initial prediction, the higher the true positive rate, but the lower the true negative rate.

VIII. Future Work

Building on our preliminary results, it may be interesting to see the outcome of a model trained with adversarial loss, on a much larger data set containing even more fonts. It would be

interesting to experiment with data augmentation, specifically artificial font skew, font dilation, and extra background noise. In order to truly validate StyleTransferOCR, we will have to evaluate our model formally on a real historical document data set with labels.

Additionally, the use of the U-Net architecture is also preliminary. One might imagine all sorts of different training schemes, models, and architectures that might improve the training. TextStyleBrush [34] is a training technique that demonstrated extraordinary performance on scene text data. This method can be tested on historical document data.
Our technique is still limited by the similarity of training data to the real historical data which we wish to learn to recognize. Therefore, future work could be to create machine learning data generators leveraging the huge amount of real documents we have collected.

To make HMMs a viable component of the pipeline, the first proposed change would be to move from an implementation of the Viterbi algorithm to an implementation of the Baum-Welch algorithm [4]. The Baum-Welch algorithm is a variant of the Expectation-Maximization algorithm that allows for estimating both the transition matrix and the emissions matrix. While the Viterbi algorithm decodes the HMM, the Baum-Welch learns uses the input observations and states to estimate the unknown parameters.

The second proposed change would be to change the scale of the algorithm from a per-letter scale, to both a per-letter and per-word scale. In the per letter implementation, the Viterbi algorithm would swap unlikely letter combinations for more likely letter combinations. This scale prohibited correcting letters that were recognised as either multiple or no characters.

Introducing a per-word scale would still retain the per-letter implementation, but extend it so that the output Viterbi result would then be run through a “per-word” Viterbi algorithm. Each word would be input into the Viterbi algorithm, and the possible states would be words within the vocabulary that are similar to the input word. The transition matrix would then contain the probability of the word following the preceding word. The emissions matrix would then contain the likelihood of the observed word is each possible word within the vocabulary. In their paper, Hladek, Stas and Juhar [5] accomplish this through a heuristic, where this likelihood is dependent on the results of several spell-checkers working in parallel.

CONTRIBUTIONS

Connor worked on every aspect of the Viterbi algorithm. In this report he wrote the Viterbi algorithm section, as well as completed editing and formatting of the whole report.

Michael performed investigations into line segmentation, and binarization, with a heavy emphasis on line segmentation. Attempted to train an easyOCR as a baseline for comparing our methodology.

Alex searched for and collected the test data. Was a difficult task which required an out-of-the-box solution. Ran experiments on the binarization process and developed a program to heuristically select the ”best” performing method. Additionally,in this paper co-wrote the introduction, performed some edits and assisted with grammar checks.

Chong investigated the use of ResNet on OCR using various popular datasets including MNIST. He wrote the attention and patch based LSTM sections and performed some edits in the report.

Chen implemented some binarization algorithm. He also wrote a text segmentation algorithm. He also implemented, trained and tested the Attention LSTM, patch LSTM as well as the Style Transfer models. He also wrote the data labelling tool.He also generated the fake data. Chen worked on every part of this report besides EasyOCR and the Viterbi algorithm.

REFERENCES

[1] J. Philips and N. Tabrizi, ”Historical Document Processing: Historical Document Processing: A Survey of Techniques, Tools, and Trends”, 2020

[2] A. Fischer, M. Liwicki , R. Ingold, ”Handwritten Historical Document Analysis, Recognition, And Retrieval - State Of The Art And Future Trends (Series In Machine Perception And Artificial Intelligence Book 89)”

[3] A.J, Viterbi, ”Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”. IEEE Transactions on Information Theory. vol.13 (2), pp. 260–269, 1967

[4] L.E. Baum, ”An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes.” Shisha, O. (Ed.), Inequalities III: Proceedings of the 3rd Symposium on Inequalities. Academic Press, 1972

[5] D. Hladek, J. Stas, and J, Juhar, ”Unsupervised Spelling Correction for the Slovak Text”, Recent Advances in Electrical Electronic Engineering 11(5):392-396. 2013

[6] I. Ben Messaoud, H. El Abed, V. Märgner, and H, Amiri, ”A design of a preprocessing framework for large database of historical documents”, 2011

[7] A. Farahmand, A. Sarrafzadeh, and J. Shanbezadeh ”Document Image Noises and Removal Methods”. Lecture Notes in Engineering and Computer Science. vol. 2202. pp. 436-440. 2013

[8] S. Ismail, S. Sheikh abdullah and F.Fauzi, ”Statistical Binarization Techniques for Document Image Analysis”, Journal of Computer Science, vol.14, pp. 23-36, 2018

[9] S. Jindal and G. Singh Lehal, ”Line segmentation of handwritten Gurmukhi manuscripts.” Association for Computing Machinery, pp.74–78, 2012

[10] A. Lamsaf, M. Aitkerroum, S. Boulaknadel, and W. Fakhri, ”Line segmentation and word extraction of Arabic handwritten text”, Association for Computing pp. 1–7. 2010

[11] J. Kumar, W. Abd-Almageed, L. Kang, and D. Doermann., ”Handwritten Arabic text line segmentation using affinity propagation”, Association for Computing Machinery, pp. 135–142, 2010

[12] N. Sakao and Y. Dobashi., ”Fonts Style Transfer using Conditional GAN”, 2019 International Conference on Cyberworlds (CW), pp. 391-394, 2019

[13] P. Krishnan, R. Kovvuri, G.Pang, B.Vassilev and T.Hassner., "TextStyleBrush: Transfer of Text Aesthetics from a Single Example", Journal of Latex Class Files, vol. 14, pp. 1-11, 2015
[14] E. Sabir, S. Rawls, and P. Natarajan, "Implicit language model in LSTM for OCR," Proc. 14th IAPR Int. Conf. Document Analysis and Recognition(ICDAR), vol. 07, pp. 27–31, 2017
[15] X. Qiu, W. Jia and H. Li, "A Font Style Learning and Transferring Method Based on Strokes and Structure of Chinese Characters," 2012 International Conference on Computer Science and Service System, pp. 1836-1839, 2012
[16] G. Atarsaikhan, B.K. Iwana, A. Narusawa, K. Yanai and S. Uchida, "Neural Font Style Transfer", Proc. 14th IAPR Int. Conf. Document Analysis and Recognition(ICDAR), vol. 07, pp. 51-56, 2017
[17] X. Zhou, Z. Zhang, X. Chen and M. Qin, "Chinese Character Style Transfer Based on the Fusion of Multi-scale Content and Style Features", 40th Chinese Control Conference, pp. 8247-8252, 2021
[18] A.K. Bhunia, A.K. Bhunia, P. Banerjee, A. Konwer, A. Bhowmick, P.P. Roy and U. Pal, "Word Level Font-to-font Image Translation using Convolutional Recurrent Generative Adversarial Networks", 2018 24th International Conference on Pattern Recognition (ICPR), pp. 3645-3650, 2018
[19] C. Li, Y. Taniguchi, M. Lu and S. Konomi, "Few-shot Font Style Transfer between Different Languages", 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 433-442, 2021
[20] L. Wu, X. Chen, L. Meng and X. Meng, "Multitask Adversarial Learning for Chinese Font Style Transfer", 2020 International Joint Conference on Neural Networks (IJCNN), pp. 1-8, 2020
[21] L. Kang, P. Riba, M. Rusinol, A. Fornes and M. Villegas, "Distilling Content from Style for Handwritten Word Recognition", 2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 139-144, 2020
[22] E. Eman, S.S. Bukhari, M. Jenckel and A. Dengel, "Cursive Script Textline Image Transformation for Improving OCR Accuracy", 2019 International Conference on Document Analysis and Recognition Workshops (ICDARW), pp. 59-64, 2019
[23] A. Zhu, X. Lu, X. Bai, S. Uchida, B.K. Iwana and S. Xiong, "Few-Shot Text Style Transfer via Deep Feature Similarity", IEEE Transactions on Image Processing, vol. 29, pp. 6932-6946, 2020
[24] X. Yu and J. Saniie, "Recognizing English Cursive Using Generative Adversarial Netowkrs", IEEE International Conference on Electro Information Technology (EIT), pp. 293-296, 2020
[25] J. Baek, G. Kim, J. Lee, S. Park, D. Han, S. Yun, S. Oh, J. Seong and H. Lee, "What Is Wrong With Scene Text Recognition Model Comparisons? Dataset and Model Analysis", International Conference on Computer Vision (ICCV), 2019,
[26] B. Jurafsky and J. Martin, "Speech and Language Processing", 2021
[27] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel and Y. Bengio, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", 2016
[28] F. Lombardi and S. Marinai, "Deep Learning for Historical Document Analysis and Recognition — A Survey", 2020
[29] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", MICCAI, 2015
[30] S. He, L. Schomaker, "DeepOtsu: Document enhancement and binarization using iterative deep learning" , Pattern Recognition, vol. 91, 2019
[31] D.P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", 2015
[32] R. Smith, "An Overview of the Tesseract OCR Engine", 2007
[33] V. Carbune, P. Gonnet, T. Deselaers, H. Rowley, A. Daryinm, M. Calvo, L. Wang, D. Keysers, S. Feuz and P. Gervais, "Fast multi-language LSTM-based online handwriting recognition", International Journal on Document Analysis and Recognition (IJDAR), 2019
[34] Krishnan, P., Kovvuri, R., Pang, G., Vassilev, B., Hassner, T. (2021). TextStyleBrush: Transfer of Text Aesthetics from a Single Example. ArXiv, abs/2106.08385.