



Lahore University of Management Sciences

Paper Unshredder

Senior Year Project Report 2020-2021

Computer Science Department LUMS

Advisor: Dr. Muhammad Fareed Zaffar

Aadam Nadeem - 21100190

Malik Ali Hussain - 21100291

Muhammad Raheem Zafar - 21100312

Acknowledgments and Dedication

In preparation for this paper, we would like to thank Dr. Fareed Zaffar for giving us the chance to work on this idea, his guidance and constant encouragement have allowed us to think out of the box. Dr. Fareed's leadership has helped us glide through the process of our research seamlessly and given us the chance to maximize our learning. We were given the liberty to try and implement any approach we wanted and even if it did not show promising results, the learning outcome helped us devise these successful strategies that are explained in the paper.

Secondly, we would thank Dr. Fareed for providing us with ample working space and resources in his lab at the Lahore University of Management Sciences. We are extremely thankful to the staff that was available in the Lab and the Computer science department who helped us throughout these two semesters.

Acknowledgments and Dedication	2
Abstract	4
Background	4
Introduction	7
Machine Learning based Approach	9
Overview	9
Dataset Generation	9
Model Training	14
Model Testing	15
Model Usage for Unshredding a Single Document	16
Limitations	19
Conclusion	19
Puzzle Approach	20
Overview	20
Pre-Processing	21
Dataset Generation	21
Processing the shreds	23
Puzzle Algorithm	24
Predicting	24
Matching	25
Results	26
Limitations	27
Future Work	27
References	30

Abstract

This paper gives an overview of two different approaches that are able to assemble shredded paper strips back into a complete document. These solutions are categorized into two parts, one machine learning-based and the other is a puzzle approach that uses a basic approach to solve the problem. The paper will explain the whole trajectory that the data takes while being cleaned and processed to be fed as training data for the machine learning model and comparison data for the basic puzzle model. Further the results of both these approaches would be discussed and would be evaluated on complexity and efficiency.

Background

Sensitive information has been documented in the form of writing since the inception of the paper. Such documents have evolved largely over the years and have changed from being handwritten to typed and now virtual. A few years back all sensitive information was stored as documents in safe facilities and still in many places the same technique is used. In this era of cloud storage and virtual document storage, there is still sensitive information that is stored as a hard copy rather than being stored on the internet. This is due to the risk of an attack and possible leak of information. Sensitive documents include anything from personal information to secret national-level information of various countries. This information is difficult and costly to guard for a long time and has to be discarded in a safe way.

A way to discard such documents in a safe manner was the need back then and as a solution, a paper shredder was introduced. The paper shredder was invented back in 1909 and was the basic machine that used to tear up paper into tiny pieces that could be discarded easily. The paper shredder evolved over time and in further advancements, it adopted more complicated and effective techniques to discard paper to an extent that no sensitive information could be retrieved back from it.

Modern shredders shred paper in three different techniques, strip-cut, cross-cut, and micro-cut. Strip-cut simply cuts paper into fine strips which are quite difficult to reconstruct if needed as the strips are very narrow and there are a lot of them which makes the sequence impossible to determine manually. Cross-cut shredder cuts the paper from two opposite sides making it even more difficult to reconstruct. The last type, micro-cut shred is the most complicated shredding technique used to discard documents to the maximum level.

With the invention of the paper shredder, people assumed that documents that were shredded were safe to dispose of in any way, this was not true as work had started to devise a method to reconstruct the document that was shredded. Such reconstruction is important in all magnitudes of document importance and sensitivity. It can help if any important document is shredded accidentally and at a national level intelligence forces around the world use this technique to squeeze out as much information as they can out of these shreds. Such an incident took place when Iranian students took over the US embassy in Tehran where they found shreds of documents that were extremely sensitive and contained information about US plans and their spies operating in the country. The students manually took their time to reconstruct the documents and gain

information about the US activity in the country. The approach they might have used must be resembling a puzzle. The reconstruction problem can be compared to a puzzle a child solves but of very high complexity as compared to that. Since then there have been numerous advancements in the techniques that have been proposed by different people around the world to reconstruct shredded documents.

DARPA (Defense Advanced Research Projects Agency) initiated a challenge where they invited computer scientists from all over the world to participate in a contest to propose a technique/algorithm that could reconstruct shredded paper. There were a number of techniques that were proposed, many are available along with their code on the internet. The majority of these techniques included machine learning practices that used to learn from the data fed and then used to evaluate the shreds provided to them as testing data. Many solutions were coded in python which used the pre-built Python libraries for machine learning models and for image processing etc. These libraries made the work easier as the computer scientists participating in the challenge could focus more on algorithms rather than coding.

There is still not a set algorithm that could reconstruct a shredded paper efficiently but it is well established that solving this problem is not impossible. This paper also proposes a solution that could potentially be an efficient solution for the reconstruction of shredded documents in the future.

Introduction

The problem that this paper is trying to solve circulates around the paper shredder. The paper presents another possible solution to reconstruct shredded paper if shreds are available. Previously proposed techniques used machine learning and neural networks to solve the problem. Those techniques involved a lot of complex algorithms and coding techniques that were very different from a layman's approach. The paper is further divided into two parts, the first part explains the machine learning approach used by us and the latter explains another solution which we would call the puzzle approach .

The first approach was further divided into a number of subparts. The first and most major part of this approach involved data collection and modification in such a way that it could be fed to the system which is in reality a pre-coded python machine learning model. Further details of the models chosen and their characteristics are discussed in the latter part of this paper. This data was collected by digitally and manually shredding pieces of paper into strips of constant sizes and these strips were then tweaked according to the needs and fed to the model as training data. Important information such as the border areas of the strips was extracted and only that was fed to the system to make it as efficient as possible. The approach gave us satisfactory results and was successful in joining the majority of the shreds correctly but still, we wanted to devise a simpler strategy that could solve the problem efficiently and also give accurate results.

The next and final strategy that we used was called the puzzle approach; this was inspired by the simple puzzle building approach that all of us have used while solving

puzzles in our childhood years. The system was fed with all possible permutations that each letter of the English language could be shredded into, these permutations were used as reference points for comparison while stitching shreds together. The shreds being processed were matched to this data and their likelihood of being each letter was calculated. The letter was decided on the basis of this likelihood. Then two shreds were compared iteratively and the ones with the same likelihood of letters were stitched together. The sample space was reduced by one on every iteration as one shred was stitched already; this reduction in sample space made the system less complex after each iteration. This approach was tested on a small dataset and provided us with promising results while using English uppercase, lowercase, and numerical digits. Complexity issues could come up when a bigger set of shreds is being processed but as the paper aims to find a simple solution so it serves the purpose. Further work can be continued which can focus on the efficiency of the solution.

Machine Learning based Approach

Overview

In modern days, Machine learning has provided solutions to almost every problem previously thought to be impossible for a computer to solve. When we approached this problem using Machine Learning algorithms, we were faced with the challenge of generating the appropriate data set for training the model. Once the dataset was generated we then opted to use a neural-network-based approach for this problem and tried both the ResNet and Squeezenet models. These models provided us with reasonable accuracy and finally after devising a sorting algorithm we were able to sequence the shreds of a paper correctly.

Dataset Generation

We looked back into previous research relevant to the project and reached a consensus that we need to train the model to distinguish between the true and false pairing and therefore should generate the training data set accordingly. A Pairing can be described as the resultant image we get after merging two shreds by placing them side by side in such a way that their edges are merged to form a single image. Pairing can be of two types(illustrated in the figure below): True Pairing, generated by placing the shreds in the correct sequence and orientation in a way like which they were in the original document, and a False Pairing is generated by merging two mismatched shreds

vertically with each other. The mismatch can be on the basis of either the wrong sequence or orientation of the shred.

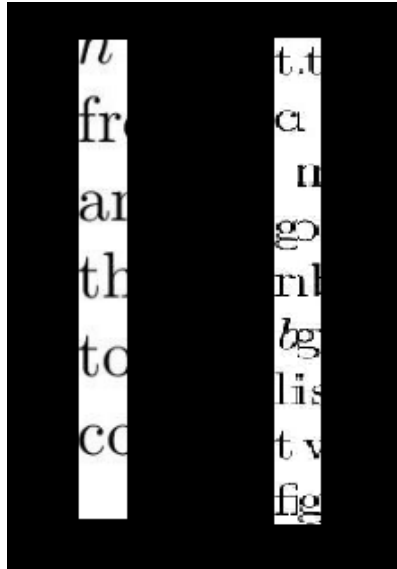


Fig: True(Left) and False(Right) Pairings

Since the original shreds had large dimensions, we decided to use only the pixels values from both the left and right edge instead of the whole shred image mainly due to two reasons. Firstly, we faced computational constraints, as our systems had at max 8 GBs of RAM and Google Colaboratory's runtime crashed several times, despite using GPU as well, whenever we tried to handle data beyond a certain size. Secondly, as we intended to make a decision on the basis of the nature of pairing, the decision can be made very efficiently by just considering a small number of pixel values from the edges of both shreds merged. This is because if the pixel values correctly align with each other to form a valid character, the model should be able to make a prediction. We did not expect our model to make sense of the contextual meaning of the text and hence the middle part of the shred became very much irrelevant and a useless data load in our

project. Therefore, we decided to only include the edges for making pairings and ignored the middle pixel values of each shred.

The algorithm we devised to generate the pairings first took two whole sherds of the original size as input. It then took the last sixteen pixels of the right edge from the left shred and sixteen pixels from the left edge of the right shred and discarded the remaining pixel values. These arrays of pixel values(each with sixteen columns) were then checked for the number of distinct or unique pixel values they had. This was mainly done because several times we encountered shreds that had an abundance of white color with very few black pixels or in other words had very few words along the axis of their cut. As such shreds had no words dissected along the cutting axis therefore when merged we got a merger of two white columns containing no possible values of non-white color to decide whether the pairing is a true one or a false one. This, if left unaccounted would have led to the generation of a large number of pairings although labeled correctly but had little to no info for the model to learn as the axis along which it was cut was simply a merger of white spaces only. Therefore, each array containing the rows of the last sixteen pixels from the edge was checked for the number of unique or distinct values. If the array contained less than 50 non-white pixel values(values having a magnitude of less than 255), it was deemed as unfit for generating a pair, and hence, the shred along with its corresponding other shred was also dropped from being the potential list of pairings to be used in data set generation. An example of such pairing can be seen below.

Once we had the meaningful pairings, the algorithm then merged two arrays containing edge pixel values into one to generate a possible pairing. The resultant pairing is an

array of length $(n \times 32)$ rows, where n is the length of the shred in terms of pixel values. This still resulted in an array too large to be given as an input to the network. So, we decided to further slice it down along the horizontal axis. We agreed upon slicing the pairing with an interval of 32 rows, thus generating the final pairings to be used as training data of size (32×32) arrays. These pairings were small enough to be given as input to the model and also large enough to yield useful information for distinguishing between a correct and wrong match as generally the font size used is way less than 32 pixels in height. However, there was still an issue of white space abundance in the smaller pairings generated after slicing the larger one. This issue occurred mainly because although we checked for the number of unique pixel values, against a threshold, in the larger shred pairing but still when it was sliced into smaller chunks, it also yielded some pairings which can be considered misleading or useless for the following reasons. Firstly, these smaller shards had an issue of white edge merger, the one mentioned earlier for the case of the larger shred pairing. But it turned out that paragraphing and several typing patterns led to the possibility of generation of such smaller pairings which had the issue of white space merger along the axis which they are merged. Moreover, another issue was a skew of information towards one side. Let's say if all the text or the words in a pairing were arranged in such a way that most of the information is either in the first sixteen columns of the array or the last ones. This left a very small sample space for the model to pick up on the differences between a match and a mismatch. So, to cater to these issues, the algorithm discarded the pairings whose 16 and 17 columns had all white pixel values, thus eliminating the issue of white space merger, and it also discarded the pairings that had less than a threshold(50 in this

case) number of distinct values in either the first or the last sixteen columns of the array. The following pictures depict the kind of sherds eliminated after this step:



Fig: Example of pairings eliminated from the dataset

With this, we were finally in a position to generate labels so before exiting, the algorithm generated the labels for each pairing to help identify the type of match: True or False. It assigned a value of '1' for True matches and a value of '0' for False matches. The resultant array of (32 x 32) pixel values along with its label was saved in a tuple containing values in the same order as described and the list of tuples generated from the pairing of two shreds is then returned to be concatenated with the original array containing all the training data points. An example of final pairings can be seen below.



Fig: Final False and True Pairing of size (32 x 32)

These strict and very minor checkmarks allowed us to not only reduce the size of the dataset drastically, thus helping us save a lot of computational power, but also helped in generating a high-quality dataset with very few possible pairings that might have led to a decrease in the accuracy of our model in the real world. After all the preprocessing, we

were able to generate a dataset containing 65124 pairings along with their labels. These were then further split into test and train data using a threshold value of 0.2. This led to the division of the dataset into 52099 training data points and 13025 test data points.

Model Training

After several meetings and brainstorming sessions, we decided to use Neural Network to solve the problem. Upon researching, we came across a few possible Convolutional Neural Networks which might fit the requirements of our project. These include various CNNs like ImageNet, SqueezeNet, and ResNet, etc. The ResNet model suited the dimensionality, time, and complexity requirements of our dataset and hence, we used the Keras module in python to use the built-in implementation of the model. The Keras implementation made it a lot easier to try and test the model with different types and levels of inputs as compared to the self-coded models.

ResNet is a Neural Network used mostly for image classification purposes and can have several weight layers ranging from 50 to 152. We tried all three of the most popularly used ResNet implementations namely: ResNet-50, ResNet-101, ResNet-152, and got the highest accuracy with the ResNet-50 model.

The model required the input to be normalized, so we normalized all the data points by dividing them by 255. Apart from this, as we had to take into account the computational constraints as well so we only ran the model for 10 epochs with a batch size of 128. As far as the parameters used for compiling the model were concerned, we used a learning

rate of 0.0001, 'Adam' optimizer, 'categorical_crossentropy' loss function, and 'accuracy' metric for training the model.

Although we were only able to run the model for 10 epochs only but perhaps due to the high quality of the data and correct hyperparameters we were able to get our model to converge to a considerably high training accuracy(in the mid 90's) and less training loss(less than 1).

Model Testing

The model's accuracy was then tested on the test dataset containing 13025 data points. We used the built-in *evaluate* function for testing the model and got a testing accuracy of 87.4% and a testing loss of 0.435. The confusion matrix for the model's prediction shown below clearly depicts that our model predicted a very high number of true positives and negatives.

```
1 preds = model.evaluate(x_test, y_test)
2 print ("Loss = " + str(preds[0]))
3 print ("Test Accuracy = " + str(preds[1]))
```

408/408 [=====] - 57s 136ms/step - loss: 0.4315 - accuracy: 0.8741
Loss = 0.431469202041626
Test Accuracy = 0.8740882873535156

Fig: Model testing results

	0	1
0	4613	1319
1	321	6772

Fig: Confusion Matrix of the model

The false predictions, although very few, were also observed. Upon, individual analysis of a random sample taken from the false predictions it was concluded that the model failed to label them correctly because they either contained very ambiguous information (like a merger of sliced '1' and 'l' etc.) or insufficient data to reach a conclusion (for e.g: the data was evenly distributed on both sides of the merger axis but not on the axis itself, except few pixel values leading to a kind of white merger issue mentioned earlier).

As the accuracy of the model was highest as compared to the others with a different number of layers, we deemed it reasonably fit for moving ahead with it in our project.

Model Usage for Unshredding a Single Document

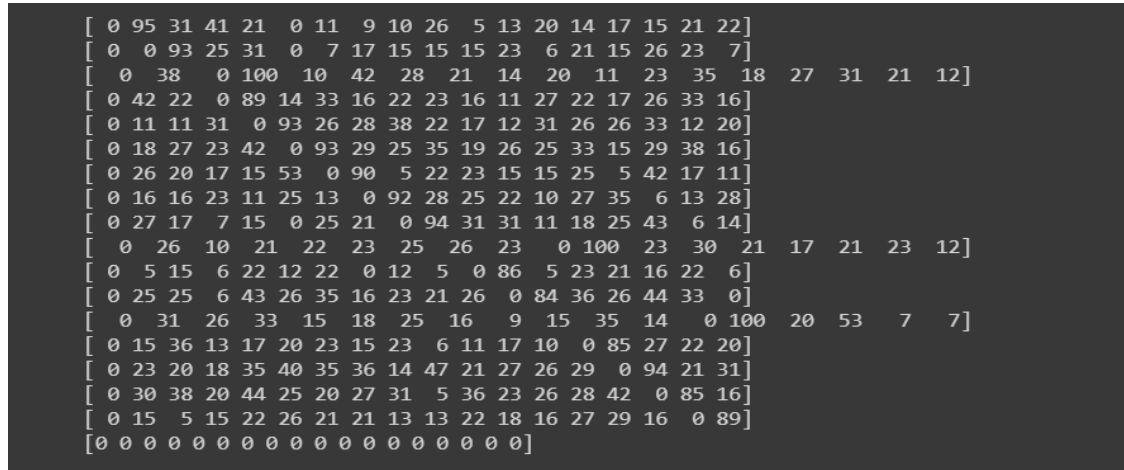
The main objective of the project was to find the optimal pair or adjacent shred for a given input shred from the whole dataset. This effectively meant that we had to somehow figure out the degree or extent to which a shred matches with another in the dataset and then choose the one with the highest. We decided to calculate the probability that given two shreds how likely they are to be a correct pairing. This could have been done in multiple ways but we opted for a more simple and comprehensible approach.

We gave the program an unknown page as input. It was shredded and then preprocessed as described earlier to generate the pairings of (32 x 32) dimensions for all possible mergers among the shreds. For each merger, all the possible pairings were

given as input to the model and based on the predictions/labels assigned by the model, the probability of a possible pairing was calculated using the following formula:

$$\text{Matching Probability}(i,j) = \frac{\text{Total number of true predictions}}{\text{Total count of pairings}}$$

This Matching Probability was calculated for a shred with all other shreds in the sample space. It was done for all shreds and a compatibility matrix, of size (n x n), comprising the values of the matching probabilities were computed. An example of such a matrix is attached below for reference.



[0 95 31 41 21 0 11 9 10 26 5 13 20 14 17 15 21 22]
[0 0 93 25 31 0 7 17 15 15 15 23 6 21 15 26 23 7]
[0 38 0 100 10 42 28 21 14 20 11 23 35 18 27 31 21 12]
[0 42 22 0 89 14 33 16 22 23 16 11 27 22 17 26 33 16]
[0 11 11 31 0 93 26 28 38 22 17 12 31 26 26 33 12 20]
[0 18 27 23 42 0 93 29 25 35 19 26 25 33 15 29 38 16]
[0 26 20 17 15 53 0 90 5 22 23 15 15 25 5 42 17 11]
[0 16 16 23 11 25 13 0 92 28 25 22 10 27 35 6 13 28]
[0 27 17 7 15 0 25 21 0 94 31 31 11 18 25 43 6 14]
[0 26 10 21 22 23 25 26 23 0 100 23 30 21 17 21 23 12]
[0 5 15 6 22 12 22 0 12 5 0 86 5 23 21 16 22 6]
[0 25 25 6 43 26 35 16 23 21 26 0 84 36 26 44 33 0]
[0 31 26 33 15 18 25 16 9 15 35 14 0 100 20 53 7 7]
[0 15 36 13 17 20 23 15 23 6 11 17 10 0 85 27 22 20]
[0 23 20 18 35 40 35 36 14 47 21 27 26 29 0 94 21 31]
[0 30 38 20 44 25 20 27 31 5 36 23 26 28 42 0 85 16]
[0 15 5 15 22 26 21 21 13 13 22 18 16 27 29 16 0 89]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Fig: An example of compatibility matrix containing Matching Probabilities generated by the program

Once the compatibility matrix was computed, the only task left was sequencing the shreds in the correct order. For this task, we used a relatively simplistic approach due to time constraints towards the end of the Fall semester. We searched for the index of the highest value in each row and then horizontally concatenated the shred corresponding to the row index with the shred corresponding to the index of the highest value in the row. We repeated this step n number of times, to make sure that we include all the

shreds in the final array consisting of concatenated shreds to form the original document back again. The final output of the program was a picture sequenced correctly with an accuracy of approximately 95% and can be seen below.

set $pk := \langle pk_1, \dots, pk_t \rangle$ and $sk := \langle sk_1, \dots, sk_t \rangle$ where each (pk_i, sk_i) is an independently generated key-pair for some one-time-secure signature scheme. The state is a counter i initially set to 1. To sign a message m using the private key sk and current state $i \leq t$, compute $\sigma \leftarrow \text{Sign}_{sk_i}(m)$ (that is, generate a signature on m using the private key sk_i) and output (σ, i) ; the state is updated to $i := i + 1$. Since the state starts at 1, this means the i th message is signed using sk_i . Verification of a signature (σ, i) on a message m is done by checking whether σ is a valid signature on m with respect to pk_i . This scheme is secure if used to sign t messages since each private key of the underlying one-time-secure scheme is used to sign only a *single* message.

As described, signatures have constant length (i.e., independent of t), but the public key has length *linear* in t . It is possible to trade off the length of the public key and signature by having the signer compute a Merkle tree $h := \mathcal{MT}_t(pk_1, \dots, pk_t)$ (see Section 5.6.2) over the t underlying public keys from the one-time-secure scheme. That is, the public key will now be $\langle t, h \rangle$, and the signature on the i th message will include (σ, i) , as before, along with the i th value pk_i and a proof π_i that this is the correct value corresponding to h . (Verification is done in the natural way.) The public key now has constant size, and the signature length grows only logarithmically with t .

Since t can be an arbitrary polynomial, why don't the previous schemes give us the solution we are looking for? The main drawback is that they require the upper bound t on the number of messages that can be signed *to be fixed in advance*, at the time of key generation. This is a potentially severe limitation since once the upper bound is reached a new public key would have to be generated and distributed. We would prefer instead to have a single, fixed public key that can be used to sign an *unbounded* number of messages.

Let $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ be a one-time-secure signature scheme. In the scheme we have just described (ignoring the Merkle-tree optimization), the signer runs t invocations of Gen to obtain public keys pk_1, \dots, pk_t , and includes each of these in its actual public key pk . The signer is then restricted to signing at most t messages. We can do better by using a “chain-based” scheme in which the signer generates additional public keys *on-the-fly*, as needed.

In the chain-based scheme, the public key consists of just a single public key pk_1 generated using Gen , and the private key is just the associated private key sk_1 . To sign the first message m_1 , the signer first generates a new key-pair (pk_2, sk_2) using Gen , and then signs both m_1 and pk_2 using sk_1 to obtain $\sigma_1 \leftarrow \text{Sign}_{sk_1}(m_1 \| pk_2)$. The signature that is output includes both pk_2 and σ_1 , and the signer adds $(m_1, pk_2, sk_2, \sigma_1)$ to its current state. In general, when it comes time to sign the i th message the signer will have stored $\{(m_j, pk_{j+1}, sk_{j+1}, \sigma_j)\}_{j=1}^{i-1}$ as part of its state. To sign the i th message m_i , the signer first generates a new key-pair (pk_{i+1}, sk_{i+1}) using Gen , and then signs m_i and pk_{i+1} using sk_i to obtain a signature $\sigma_i \leftarrow \text{Sign}_{sk_i}(m_i \| pk_{i+1})$. The actual signature that is output includes pk_{i+1} , σ_i , and also the values $\{m_j, pk_{j+1}, \sigma_j\}_{j=1}^{i-1}$. The signer then adds $(m_i, pk_{i+1}, sk_{i+1}, \sigma_i)$ to its state. See Figure 12.4 for a graphical depiction of this process.

Fig: Final output from the Paper Unshredder

Limitations

As per our understanding, our program can under-perform under few circumstances.

Firstly, the trained model might not work with higher accuracy on languages with more complex and relatively denser(in terms of non-white pixel values) notation. An example of such a language is Mandarin Chinese. The reason behind this problem is mainly the small size of our dataset which resulted in lesser diversity among the data points. We wanted to increase the size of the dataset for incorporating more diversified possible matches but unfortunately, we were restricted by the computational capacity of our personal computers.

Secondly, we also acknowledge that the algorithm used for sequencing the shreds is not the most time and space-efficient and therefore, will cause problems while scaling this up for several pages long documents. This is because the algorithm used right now is of the order $O(n^2)$ and will eventually become NP-hard for larger datasets. Therefore, we propose the use of a more efficient graph-based solution for the problem which is explained in the following section.

Conclusion

After successfully Unshredding a complete one-page document we can say that we were able to achieve the goals we set at the beginning of the semester to a reasonable extent. However, it is worth mentioning that we managed to complete this project in the difficult semesters we experienced in LUMS. This was mainly because of the challenging circumstances due to covid-19. The new mode of communication made it a

lot more difficult to find a mutually acceptable meeting or working slot among the group mates and also with the project advisor. This was compounded by the ongoing grad school application cycle and the typical semester workload as well, thus making it more challenging to complete the project within the previously decided timeline. However, due to our unwavering commitment and constant guidance from the project advisor, Dr. Fareed Zaffar, we were able to complete the project in its entirety but for a smaller problem set. We are looking forward to working on the project over the summer break for achieving the remaining goals, mainly relevant to the scaling up of the project as described above in the Future Work section.

Puzzle Approach

Overview

Upon successfully reconstructing a shredded document through a neural network based approach, we took upon ourselves the challenge, advised by our supervisor, to solve the problem at hand through a much more simplified approach. This approach was more similar to constructing a puzzle from its broken pieces using a miss-and-match algorithm. Like pieces of a puzzle, we had processed shreds of paper from a shredder. The goal was to successfully realign each shred by matching the correct shred from the right and the left side. In this approach, the essential sub-problems we aimed to solve were to propose a fully automatic approach to the preprocessing, feature extraction, and matching phases such that human involvement could be minimized, and developing a

rotation technique to solve the problem through a geometrical algorithm with the use of computer vision.

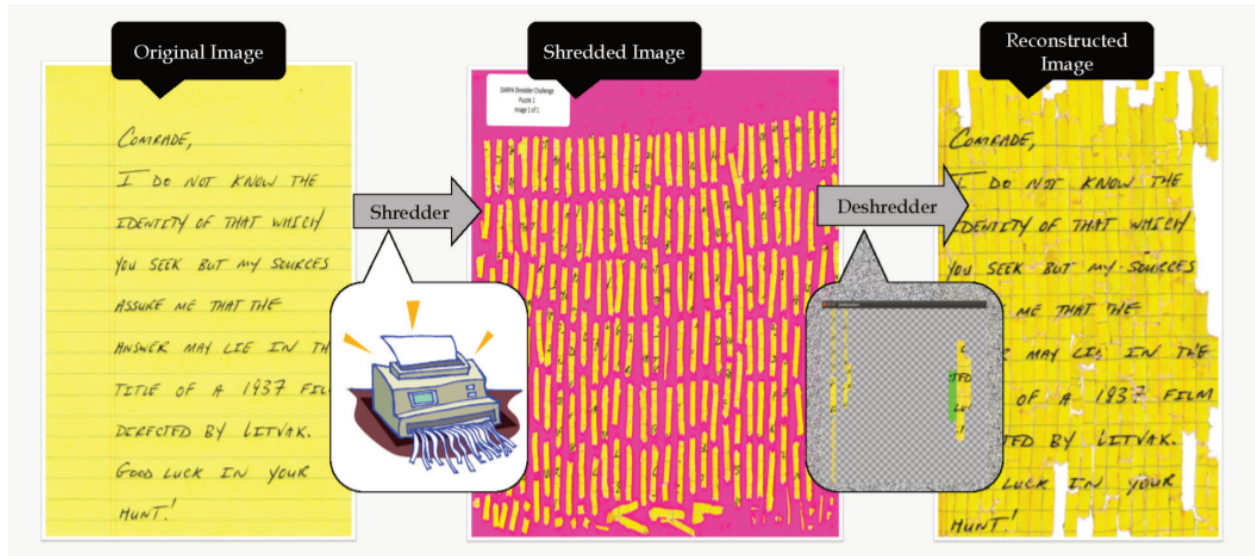


Fig: Overview of Unshredder. (Left) original document; (Middle) Document shreds to be reconstructed; (Right) image reconstructed from the shredded pieces

Pre-Processing

Dataset Generation

An algorithm classifies an action to be either true or false by comparing it with a fixed answer. In our case, we would want our algorithm to correctly identify letters of the sides of a shred to successfully pair it with its corresponding shred. Our algorithm takes 2px from each side of a shred and matches it with 2px of each side of the other shreds. This required us to identify a letter from those 2 pixels and match it with what we find the best result from the other shreds. To do so, we generated a dataset of all the letters (uppercase and lowercase) and numerical of the font 'Times New Roman' by slicing

them in every possible direction. The method included taking a single 46x46 pixel image of a letter and coming up with all possible permutations (vertically and horizontally) in which it can be sliced from a shredder. Our dataset ignored complete white shreds of a letter and discarded them as they were only misleading the algorithm. This resulted in each letter having a different number of variations. Below is an example of some of the possible slicing of the letter A after it was processed.

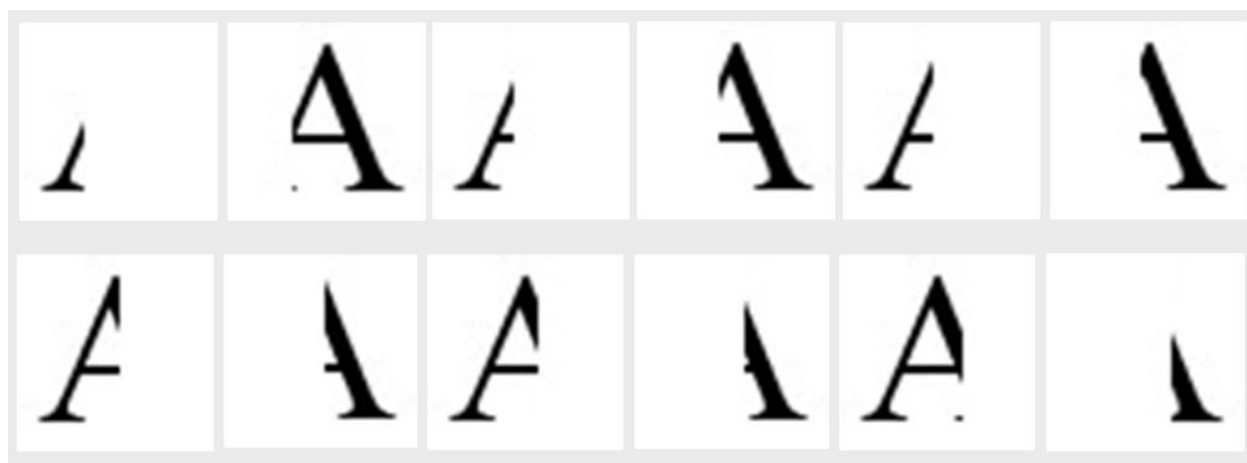


Fig: Vertical slicing of the letter 'A'

Here you can see that using the vector image of a simple letter 'A', vertical slicing of 2px each sideways and horizontal slicing of 2px each from the top and bottom were performed a large number of times to not leave any room for error. On the basis of how our algorithm takes segments of the shreds, horizontal shredding was needed. There existed no fixed amount of space to skip for white spaces when choosing rectangular pieces of the sides of the shreds and often each shred contained lines of more than one letter in it. Horizontal shredding added to the dataset similar alphabets that were cut from the top to the bottom. This led to a dataset of around 6760 possible Times New Roman alphanumeric images for the algorithm to match from. Vectorization was

performed on the images to convert them into equal-sized matrices with a value in each position ranging from 0 to 255 where 0 indicates black and 255 indicates a white space, to facilitate the puzzle algorithm.

Processing the shreds

Accurate artificial shreds were used in the preprocessing section. They were designed in a way to clone real shreds from a paper-shredder machine. A single A4-sized page was used in the process and a simple python script was used to cut it into 20-30 shreds vertical. Some noise was added to each shred to mirror it as close as possible to an actual shred. Each shred is then processed separately where it goes through a sequence of steps. The first step involves a thin strip of size 46 pixels from the left and right side of the shred to be selected. This strip is then divided into smaller, rectangular segments of length 23 pixels and width 46 pixels each. Doing so gives us an approximate measure of each edge of a line of the paper. After ignoring completely white segments, we get a collection of rectangles each with some sequence of letters. The final step includes slicing 2 pixel thin vertical shreds from the sides (right if the rectangle was taken from the right side of the shred, and left if it was taken from the left side). These shreds are essential for the algorithm as they are used to predict the matching sides of shreds. Similar to the dataset generation, these shreds were also stored as matrices of equal sizes with a value in each position ranging from 0 to 255 where 0 indicates black and 255 indicates a white space.

Puzzle Algorithm

Predicting

The preprocessing section has designed the data in a way that makes it ready to use with our algorithm. As explained before, the algorithm works in a way similar to reconstructing a puzzle, you analyze a piece and look for its best fit among the options available. Similar to the idea, as we have generated vectors of the dataset of each letter and number, as well as vectors of the given minimal sheet of paper, we can process these to match the shreds. The execution takes place by first getting each thin shred of the processed incoming test data, converts it into a vector, and sends it to a function. The idea of the function is that it takes a single vector and finds the best possible match of the line(s) in it by comparing it with each vector element letter from the dataset. The comparison takes place between the vectors of the dataset and the shred to be examined.

The mechanism is as follows, each position in a vector is examined and compared with that of the dataset letter vector and classified as true if it is identical and false if it does not come close. The criteria of classifying it as true or false have been altered and played around a couple of times to get different accuracies. Assumptions were taken into account where the color of the lines in the incoming shreds might not be the most accurate due to a number of reasons including noise, scanner quality, etc. which might not produce similar vectors for the same set of lines in the dataset. Hence, an offset of ± 5 was added to the vectors to increase the number of positives and in turn, increase the accuracy with a letter from the dataset. These accuracies of the vector with each

letter were stored in an array and sorted such that the top five with the highest accuracies are displayed. The letter with the highest accuracy and a frequent number of appearances in the top five was selected as the prediction. This same step was done for all vectors of the shredded paper and predictions were then compared with the actual labels to determine the accuracy of the algorithm. The accuracy turned out to be 75% for the provided test document.

Matching

Predictions of the top five letters of each side of the 2 pixel-wide rectangular shreds are now stored in the program and it is our job to match it with its corresponding pair. The process of matching involves comparing the calculated top five predictions of the right side shreds with the predictions of the left side shreds and vice versa. Consider a strip from one of the right-sided shreds, our algorithm will have to compare it with its corresponding strip from the left-sided shreds. If there exists a significantly similar letter in both their predictions, those shreds are said to be a pair with each other.

Matching set: [predictions from left side] \cup [predictions from right side]

The criteria for matching take into account the number of unique predictions, the accuracy of those predictions, and the intersection of two sets of predictions with each other. For a better understanding of the criteria, we use an example of a set of predictions from a right shred, $[M, M, N, N, A]$. This means that the top five predictions in descending order of their accuracies are M, M, N, N , and A . The unique values in the chosen set are M, N, A . Their prediction accuracies could vary by a lot or by none, M may have an accuracy of 99% while A may have an accuracy of 30%. These will help

us in the final decision if we have a significant overlap of final matches. Now consider an example of a set of predictions from the left shred, $[M, N, W, W, M]$. The unique values in this shred are M, N, W . After we have the two sets of predictions, an intersection is taken among them. The elements in the intersection lead to the final matching prediction. For the two sets chosen above, the intersection of their unique variables results in the set $[M, N]$. Our initial predictions have been narrowed down to two letters, one of which is the correct one. To choose a single letter from the set in the intersection, the positions of each letter in their initial sets are taken into account. As per our example, since M appears earlier than N , it is automatically given a higher preference based on its prediction accuracy. This same algorithm is used with all corresponding shreds and results in matching pairs that lead to stitching the shreds back up to reconstruct the shredded document.

Results

Designing and streamlining the process of generating shreds, predicting letters, and matching sides lead us to check the algorithm on a sample set of A4-sized papers. Following the process, we determine the shreds and store them as vectors according to the right and left sides. We run the algorithm on the shreds and determine the predictions and the accuracy of those predictions. The accuracy comes out to be around 75% on the test data and more than 98% on our initial data. Matching these shred predictions with the corresponding predictions of the shreds, we gather all the correct matches and stitch the shreds with each other accordingly. The initial and final

versions of the sample image turned out to be identical and proved the success of our algorithm.

Limitations

Although the algorithm works accurately on our sample paper, it is subjected to a few limitations as well. One of these includes the limitation of the font and font size on the paper. As our whole generated dataset comprises letters and numbers of the same font (Times New Roman) and font size (12 pixels), it will not do a good job while trying to predict letters of any other style. Similarly, if the paper contains too many images, the algorithm will try to match that with the letters and numbers from the dataset and will face a hard time. While the matching section of the algorithm may try to match images with each other, the accuracy is not promised. Another limitation includes too much whitespace on the sides and in between the paper. This causes the algorithm to mistakenly misalign only one shred (the most left one) with the other (the most right one) due to the matching algorithm matching the whitespaces together.

Future Work

The work done up till now was done as a final year undergraduate project. Both the approaches used gave us promising results and both in their own domains have a lot of potential to be viable solutions.

As explained above there were decent levels of accuracy shown by both models. We believe that if worked and researched in more depth, these models can prove to be the

basis of a solution that actually re-assembles shredded documents in a non-complex efficient manner and can be used at scale.

For the first approach, future work will mainly be focused on increasing the accuracy and efficiency of the system and scaling it up for handling larger and more diverse documents. The plan is to work on collecting exhaustive data to feed into the machine learning model as training data. This will require a lot more time and effort as a balance has to be maintained between overfitting and underfitting the model as well as maintaining accuracy. The collection and preprocessing of the data would be the greatest challenge in working on this approach as data volume would be much higher than what we used now.

As far as the accuracy and efficiency of the program are concerned, we expect to have access to the Computer Science department's supercomputer this summer. This will allow us to process and thus generate a very large dataset as compared to the current one. With a considerably larger dataset and computational power at our disposal, we believe we can train a better model with more hidden layers and a lot quicker than the one generated on our personal computers. This will surely reduce the training time as well as the testing accuracy and general usability of the model for various types of notations and languages.

Moreover, for scaling up the program to cater to multiple page input, we need to work on making the matching probability calculations and the sequencing algorithm more efficient. The matching probability calculations can only be scaled up by using either super or distributed computers and since we intend to use one of them this summer, we

hope that this issue will be resolved to a great extent. For the sequencing algorithm, the current algorithm should be replaced by a graph-based one to reduce the time and space complexity. One possible proposal is the Asymmetric Salesman Problem algorithm; it can be used to find the Hamiltonian circuit in the graph and thus find the cycle much more quickly and efficiently. This algorithm is one of the very complex graph-based algorithms and hence its implementation in the Google OR-Tools can be used to find a possible way of incorporating it in the solution. Since we did not have time to do ample research on it, we are still unsure that by what factor this solution can improve the efficiency of the existing one.

For the second approach, the main area to work in is the time and storage complexity of the algorithm. An analysis will be carried out in which the algorithm will be broken down into basic steps and the number of iterations and comparisons will be tried to be reduced. This approach does not use any machine learning model but we would be trying to incorporate a model in this approach too. As data is specific and more processed in this approach we expect the model to give us better results here.

Apart from this, we also intend to account for the skewness of input shreds in both approaches. This basically means that if the input shred is tilted or at an angle rather than being absolutely upright, then the program should be capable of reorienting it in the optimal position such that all the text on the shred is upright and in correct orientation. As of now, we do not have any set algorithm in consideration and are open to adopting and working on any technique which we find reasonable when we resume working in summers.

References

Butler, Patrick, et al. *The Deshredder: A Visual Analytic Approach to Reconstructing Shredded Documents*. Department of Computer Science and Discovery Analytics Center, Virginia Tech, Blacksburg, VA 24061.

Paixao, Thiago M., et al. *Fast(Er) Reconstruction of Shredded Text Documents via Self-Supervised Deep Asymmetric Metric Learning*.

Peng Li, Xi Fang, Lianglu Pan, Yi Piao, Mengjun Jiao, "Reconstruction of Shredded Paper Documents by Feature Matching", *Mathematical Problems in Engineering*, vol. 2014

Ranca, Razvan. *Reconstructing Shredded Documents*. 2013.

Venkatesh, Vasudha. *SHREDDED DOCUMENT RECONSTRUCTION*. Dec. 2017.

Y. Liu, H. Qiu, J. Lu and Y. Fang, "Shredded Document Reconstruction Based on Intelligent Algorithms," 2014 International Conference on Computational Science and Computational Intelligence, 2014

Python Notebooks:

- [ML Approach I](#)
- [ML Approach II](#)
- [Puzzle Approach](#)