SHREDDED DOCUMENT RECONSTRUCTION

CS 297 Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS297

By

Vasudha Venkatesh

December 2017

TABLE OF CONTENTS

I. Problem Statement

Sensitive documents are usually shredded into strips before discarding them. Shredders are used to cut the pages of a document into thin strips of uniform thickness. Various techniques were developed in the past to shred top secret documents. Defense Advanced Research Projects Agency (DARPA) has been interested in techniques to assemble shredded documents and in 2011 DARPA organized the shredded document reconstruction challenge because it is a daunting job to reconstruct shredded documents confiscated from the enemy lines. The challenge also served an additional purpose of gauging the security of existing shredding mechanisms. This project aims to re-assemble shreds from multiple pages of a document using photographs of the individual shreds. Each shredded piece in the collection bin can belong to any of the pages in a document.

The task of document reconstruction involves two steps:

1. Identifying the page to which each shred belongs
2. Rearranging the shreds within the page to their original position.

The difficulty of the reconstruction process depends on the thickness of the shred and type of cut (horizontal or vertical). Thickness of the shred is directly proportional to the ease of reconstruction. Horizontal cuts are easier to reconstruct because sentences in a page are intact and not broken. Vertical cuts are harder because there is very little information to glean from each shred.

When the techniques for re-assembly are perfected, an Android app will be developed to reconstruct the pages of a shredded document by using photograph of the shreds as input. There will be no prior knowledge of the page to which each shred belongs and the thickness of each shred will conform to the measurements of a standard strip shredder. The type of shredder cut will be vertical. This project is intended to enhance existing work which currently reconstructs up to ten images from a mixed bag of square pieces.

To achieve the goal of document reassembly, the work for this semester is split into four deliverables. These four deliverables helped in understanding the applicability of the existing

work which reassembles square image puzzles for written document reassembly. Deliverable 1 focused on the mechanics of android app activity communication. Deliverable 2 focusses on evaluating the performance of the existing solver algorithm on custom images. The goal of this deliverable was to evaluate performance on images with uniform contrast values dispersed throughout the image. Deliverable 3 implements Paikin Tal placer in Java. Deliverable 4 evaluates a Recurrent Neural Networks (RNN) architecture for identifying the next best piece in a sequence.

II. Deliverable 1 – Inter-activity communication

The first step to building an Android app is to learn the architecture and model of message passing. The following terms are necessary to understand the basic architecture of an Android app:

Activity: As the name indicates, an "Activity" is what a user does with the app. Every "Activity" is displayed to the user in a window. These can be full-screen windows, floating windows or windows nested inside another window.

Intent: It represents the action to be performed. It acts as a bridge between activities. An "Intent" holds information about the action that needs to be performed by an "Activity".

The final goal of the project is to capture image using camera app and pass the "Intent" to a new activity to handle the de-shredding operation. This deliverable was intended to help understand the structure of a typical android app which performs inter-activity communication. Figure 1 shows the user interface (UI) of the app.

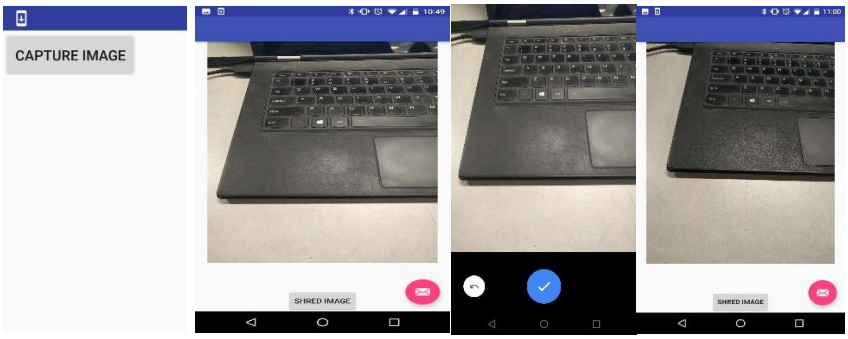

Fig. 1: App interface for Inter-activity communication

The application works as follows: The user is initially shown a screen with the “Capture Image” button which when clicked, creates a camera intent which is passed to the camera app to aid in capturing the image. The way the intent is passed is illustrated in Figure 2.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // getPermission(Manifest.permission.CAMERA);
    getPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE);
    setContentView(R.layout.activity_image_capture);
    final Activity activity = this;
    this.imageView = (ImageView)this.findViewById(R.id.ImgPhoto);
    Button photoButton = (Button) this.findViewById(R.id.BtnSelectImg);
    photoButton.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {

            Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
            if (cameraIntent.resolveActivity(getPackageManager()) != null) {
                // Create the File where the photo should go
                Uri photoFile = null;
                try {
                    photoFile = FileProvider.getUriForFile(activity,
                            BuildConfig.APPLICATION_ID + ".provider",
                            createImageFile());
                } catch (IOException ex) {
                    // Error occurred while creating the File
                    Log.i(TAG, "IOException"+ex.getMessage());
                }
                // Continue only if the File was successfully created
```

Fig. 2: IMAGE_CAPTURE intent passing to camera app

Once the image is captured, it is saved to a location and this information is passed to the shred image activity. This is the crux of inter-activity communication. The split image activity picks up the information about the file location from the capture image activity and performs the necessary operations. The message passing portion of the app is shown in Figure 3.

```
//Get the location of the image from the ImageCaptureActivity and display here
Bundle extras = getIntent().getExtras();
if (extras != null && extras.containsKey("KEY")) {
    String imagepath= extras.getString("KEY");
    Uri imageUri = Uri.parse(imagepath);
    File file = new File(imageUri.getPath());
    try {
        InputStream ims = new FileInputStream(file);
        imageViewsplit.setImageBitmap(BitmapFactory.decodeStream(ims));
    } catch (FileNotFoundException e) {
        return;
    }
}
```

Fig. 3: Communication between ImageCaptureActivity and SplitImageActivity

A particular challenge that was faced in this exercise was in figuring out the mechanism for granting permissions to the app to access the file storage system.

III. Deliverable 2 – Solver performance on custom images

The existing solver uses a hierarchical clustering mechanism for assembling a mixed bag of puzzles. This mechanism is used to assemble multiple puzzles without knowing the count of puzzles to solve. The images that were used for mixed bag assembly were fetched from the base papers and most of the images were high contrast pictures with several distinct regions that can be segmented into chunks with relatively higher level of accuracy. However, a typical grayscale document does not have many distinct regions. The contrast level is almost uniformly distributed throughout the image. These two factors pose some challenges to the existing model. The purpose of this exercise is to figure out the specific portions of the algorithm that are the areas of concern.

The algorithm for placement is as follows:

Jumble individual pieces from across puzzles: Every puzzle is split into multiple pieces and the pieces from across individual puzzles are jumbled so that a mixed bag of puzzles is created.

Calculate piece distance and asymmetric mutual compatibility: Based on the puzzle type which is either type 1 (with no piece rotation) or type 2 (with piece rotation), possible neighbors for each piece are identified and a distance matrix is created for each piece with the value at position *(i,j)* of the matrix representing the distance value between the said piece and piece *j* along the side *i*. Since the pieces are square, there are four different values for *i*. The asymmetric mutual compatibility is calculated using the formula in Figure 4.

$$D(p_i, p_j, right) = \sum_{k=1}^{K} \sum_{d=1}^{3} \|([2p_i(k, K, d) - p_i(k, K-1, d)] - p_j(k, 1, d))\|$$

Fig. 4: Distance between pieces i and j

In the above figure, d = 1 to 3 representing the color dimensions of the piece. Instead of the usual RGB color scheme, the LAB color scheme was used where L – Lightness, A and B represent green-red, blue-yellow respectively. The significant advantage of using LAB scheme over RGB color scheme is that LAB scheme is device-independent. The above formula calculates the gradient between the pixels on a particular side and the penultimate side. This gradient is the predicted value for the neighboring side on a piece j. This predicted value is compared to the actual value of piece j along the neighbor side and the piece with the lowest difference value is the predicted neighbor.

Asymmetric mutual compatibility is used to calculate the compatibility between pieces i,j as well as between pieces j,i.

$D(p\_i, p\_j, right) \neq D(\tilde{p\_j}, p\_i, left))$.

The use of asymmetric dissimilarity helps when there are missing pieces.

Calculate best buddies: As the name indicates, it is the calculation of the best possible neighbor for a piece. In order to calculate the best buddy for a piece, it is important to calculate the compatibility function.

The purpose of the compatibility function is to make sure that the pieces are truly adjacent as opposed to mere comparison with smooth areas which give misleadingly small values. This is to eliminate false positives in compatibility. The formula is shown in Figure 5.

$$C(p_i, p_j, r) = 1 - \frac{D(p_i, p_j, r)}{secondD(p_i, r)}$$

Fig. 5: Mutual compatibility metric

Piece Placement: The original algorithm for single puzzle suggested by Paikin and Tal is used in the mixed bag solver too for piece placement. A list is created with candidate for first piece placement. A piece should have best buddies on all four sides and each of those pieces must have best buddies on all their four sides too. Such a piece is placed in a queue for possible placement.

Segmentation, Stitching and hierarchical clustering: Identify segments (sets of pieces with highest placement confidence). Stitch related segments using hierarchical clustering. Each segment cluster represents a single ground-truth. The total number of segment clusters should match the total number of puzzles. This is to account for unknown number of ground-truth input puzzles. A distinct seed piece is selected from each segment cluster and the placer algorithm from Tal & Paikin is used for final piece placement.

Upon running the algorithm with a document page divided into square pieces with piece rotation allowed, the results are as shown below:

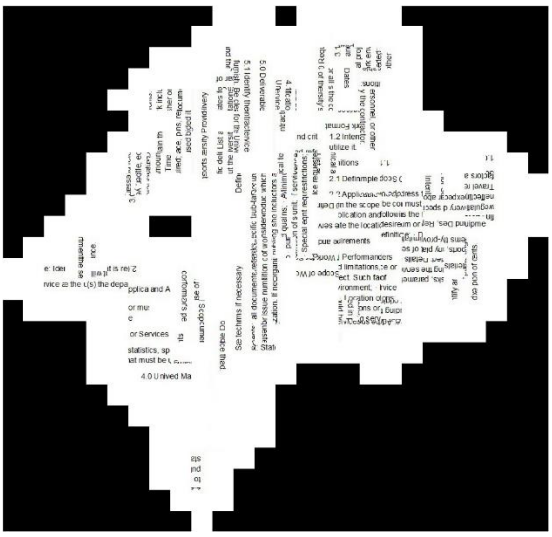

Fig. 6: Hierarchical clustering on a document with square pieces

It can be seen that with a grayscale image of text, the algorithm's performance is significantly lower than when a high contrast image of vivid colors are used.

The way the image is segmented can be seen in Figure 6 and 7.

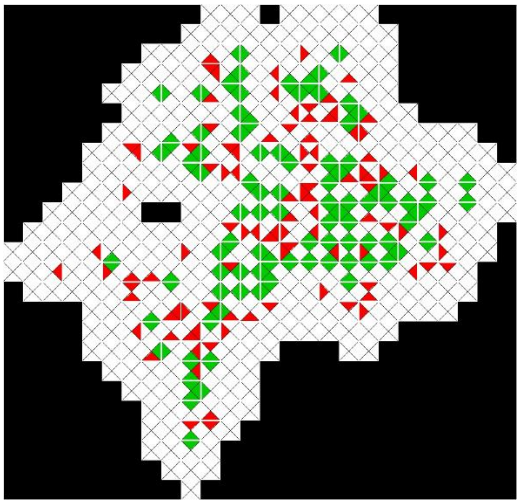

Fig. 7: Segmentation of chunks

It can be seen that the main reason the solver performance is underwhelming for documents is that there are no strong segments. This is because the piece distances are more or less uniform throughout the image.

The potential pitfalls identified from this deliverable are as follows:

1. The use of gradients for piece distance calculation results in several pieces of same distance value, hampering the accuracy of best buddy estimation.
2. Weak segmentation for images of text.

The next step involves implementing the placer algorithm in java to substantiate the hypotheses above.

IV. Deliverable 3 – Paikin Tal Placer implementation in Java

The previous exercise helped in identifying some of the weak spots of the existing model with respect to the hierarchical segmentation. The purpose of this deliverable is to implement the placer algorithm in java and test the efficacy of the placer algorithm on document images. The reasoning behind this exercise is that, irrespective of the algorithm used for solving the puzzles, Paikin Tal placer is the algorithm which is ultimately used for the final placement of pieces.

Hence, as part of this exercise, this algorithm was implemented in java. The overview of the algorithm is given in Figure 8.

Algorithm 1 Placer

```
While there are unplaced pieces
    if the pool is not empty
        Extract the best candidate from the pool
    else
        Recalculate the compatibility function
        Find the best neighbors (not best buddies)
    Place the above best piece
    Add the best buddies of this piece to the pool
```

Fig. 8: Paikin Tal Placer Overview

The code snippet which calculates the piece distance, asymmetric compatibility and best buddies is given below in Figure 9.

```
                //Calculate the distance between pieces i and j for every neighbor side
                for(int sideid_j = 0; sideid_j < neighbor_sides.length; sideid_j++)
                {
                        if(p_i == 0 && p_j==1)
                        {
                                //System.out.println("Piece is "+p_i);
                                //System.out.println("Side is "+sideid_i);
                                //System.out.println("Comparing piece is "+p_j);
                                //System.out.println("Neighbor side is "+neighbor_sides[sideid_j]);
                                //System.out.println();
                        }
                        double distance = calculateDistance(this.pieces.get(p_i), sideid_i,
eces.get(p_j),neighbor_sides[sideid_j]);
                        //System.out.println(distance);
                        piece_distance_matrix[sideid_i][p_j][sideid_j] = distance;

                        //update min and second min distances
                        if(distance < min_distance_list[sideid_i])
                        {
                                second_distance_list[sideid_i] = min_distance_list[sideid_i];
                                second_min_piece_id[sideid_i] = min_piece_id[sideid_i];
                                min_distance_list[sideid_i] = distance;
                                min_piece_id[sideid_i] = p_j;
                        }
                }
        }


//Store the distance matrix in the piece's object
this.pieces.get(p_i).setInter_piece_distance_matrix(piece_distance_matrix);
this.pieces.get(p_i).setMin_distance(min_distance_list);
this.pieces.get(p_i).setSecond_best_distance(second_distance_list);
this.pieces.get(p_i).setBest_buddy_candidates(min_piece_id);
```

Fig. 9: Paikin Tal Placer

After implementing the Paikin Tal placer, the algorithm was run on a regular image with color components and the results were compared to running the same algorithm on documents.

Figure 10 shows the results of running the placer on colorful image.

```
Minimum distance matrix is
[1445.0, 1292.0, 183.0, 739.0]
Minimum piece id is
[1, 48, 1, 10]
Asymmetric matrix is
[[[1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [
Finished Asymmetric Compatibility
Identified Best Buddy Candidates for 100 pieces
Finished Mutual Compatibility
Best buddy candidates for piece 0 is [98, 89, 42, 32]
```

Fig. 10: Placer results on colorful images

Figure 11 shows the results of running the placer on black and white document images.

```
Minimum distance matrix is
[2410.0, 0.0, 0.0, 513.0]
Minimum piece id is
[1, 1, 1, 10]
Asymmetric matrix is
[[[1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [1.0], [
Finished Asymmetric Compatibility
Identified Best Buddy Candidates for 100 pieces
Finished Mutual Compatibility
Best buddy candidates for piece 0 is [0, 1, 1, 0]
```

Fig. 11: Placer results on black and white document images

It can be observed that there is an absence of variance in the distance values. This causes the best buddy values to be chosen through the tie-breaker mechanism more often than it should be. This is illustrated in Figure 10 which is the result for best buddy values for a piece with piece id 0.

```
Best buddy candidates for piece 0 is [98, 89, 42, 32]

Best buddy candidates for piece 0 is [0, 1, 1, 0]
```

Fig.10: Best buddy variances

V. Deliverable 4 – Neural Networks for puzzle reassembly

The previous deliverable established that when it comes to documents, due to uniform distribution of contrast and lack of color, the clustering algorithm does not perform to satisfactory levels. One of the main reasons for this was the metric for piece distance calculation which was the Achille's heel of the placer algorithm. Hence, at this point, the following option was considered:

1. A neural network model utilizing LSTM for puzzle reassembly

Long Short Term Memory (LSTM) is an architecture of Recurrent Neural Networks (RNN) which uses cells determining how much of the past information is to be carried forward in the time steps forward. LSTM is well suited for puzzle reassembly because it is essentially predicting the next object in the sequence. Puzzle assembly is predicting the best possible next piece.

The model focusses on building a sequence of pieces. A sequence to label model works better than feed-forward model like CNN. LSTM can be used to model sequences with recurrent connections. It is found to be more accurate than CNN. The learning rate of the earlier layers were very less and hence the earlier convolution filters faced a difficult time finding out the edges in the jumbled up images. The architecture is as shown in the following Figure 11.

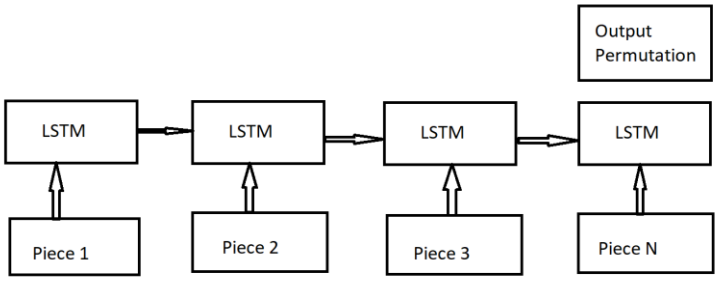

Fig. 11: LSTM architecture for puzzle reassembly

Setup of the model:

- Vectors for each of the component images (x1, x2, x3, ..., xn) are taken as different time steps of the LSTM.
- It is a many-to-one-model.
- Output is a one-hot representation of the permutation that generates the best possible shred sequence.
- The different image parts at the different time steps allowed the model to recognize the different constituents of the image.

Softmax loss is used. If the total number of pieces is 4, there are 4! = 24 possible permutations, one for each class. While testing, the softmax output of 24 classes is checked for max value. The element with the highest value is chosen as the final permutation.

VI. Conclusion

This foundation work needed for shredded document reconstruction has been completed in this project. The existing techniques and algorithm performs well on regular images. The four deliverables helped understand the applicability of the algorithm in its existing form for shredded documents. Deliverable 1 gave a hands on learning experience with building android apps which will be used eventually in CS298 work. Deliverable 2 and 3 helped identify the areas of improvement. A new model for de-shredding was tried in Deliverable 4. The knowledge gained from the CS297 semester helped in understanding the applicability of the existing algorithm for document images. The potential weaknesses about the existing algorithm learned during the CS297 work for this semester will be used as the basis for CS298 in implementing a model which reconstructs shredded documents.

References

[1] Z. Hammoudeh, “A Fully Automated Solver for Multiple Square Jigsaw Puzzles Using Hierarchical Clustering,” Thesis, 2016.

[2] G. Paikin, A. Tal, "Solving Multiple Square Jigsaw Puzzles with Missing Pieces", IEEE Conference on Computer Vision and Pattern Recognition, 2015.

[3] Lin HY., Fan-Chiang WC. (2009) Image-Based Techniques for Shredded Document Reconstruction. In: Wada T., Huang F., Lin S. (eds) Advances in Image and Video Technology. PSIVT 2009. Lecture Notes in Computer Science, vol 5414. Springer, Berlin, Heidelberg.

[4] Y. Liu, H. Qiu, and J. Lu, “Shredded Document Reconstruction Based on Intelligent Algorithms,” in International Conference on Computational Science and Computational Intelligence (CSCI), 2014.

[5] J. Pearl, M. Diem, F. Kleber, “Strip shredded document reconstruction using optical character recognition,” in International Conference on Imaging for Crime Detection and Prevention 2011 (ICDP 2011).

[6] L. Zhu, Z. Zhou, D. Hu, “Globally Consistent Reconstruction of Ripped-Up Documents,” in IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 30, Issue: 1, Jan. 2008).