Spring 2018

Android de-Shredder App

Vasudha Venkatesh
San Jose State University

Android de-Shredder App

A Project Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment of

the Requirements for the Degree

Master of Science

By

Vasudha Venkatesh

May 2018

SAN JOSÉ STATE UNIVERSITY

The Undersigned Thesis Committee Approves the Thesis Titled

Android de-Shredder App

By

Vasudha Venkatesh

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

_____

Dr. Chris Pollett, Department of Computer Science 05/07/2018

_____

Dr. Melody Moh, Department of Computer Science 05/07/2018

_____

Dr. Shruti Patil, Google LLC 05/07/2018

ABSTRACT

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. Each shredded piece in the collection bin could belong to any of the pages in a document. The task of document reconstruction involves two steps: Identifying the page to which each shred belongs and 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). The 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. In this project, an Android app is developed to reconstruct the pages of a shredded document by using a photograph of the shreds as input. However, no prior knowledge of the page to which each shred belongs is assumed. The thickness of each shred should conform to the measurements of a standard strip shredder. The type of shredder cut is vertical. This work is an enhancement of an existing work of puzzle reconstruction developed by Hammoudeh and Pollett. Through the experiments conducted on both the existing model and the proposed model, it was found that the proposed pixel correlation metric model performed with 80 to 90% better accuracy than the existing RGB metric model on grayscale document images. However, the performance on high contrast images remained almost the same at 90% accuracy for both the RGB model and pixel correlation metric model.

ACKNOWLEDGMENT

I would like to thank my project advisor Dr. Chris Pollett for his excellent guidance which helped steer this project in the correct direction.

I am also thankful to my committee members Dr. Melody Moh and Dr. Shruti Patil for their valuable time and input which enriched the outcome of this project.

I would like to thank the Department of Computer Science for providing the necessary resources to conduct this project defense in a smooth manner.

Special thanks to my husband for never giving up on me and standing like a pillar of moral support. Special love to my kid who adapted to my schedule and remained flexible.

Finally, but in no way lesser, I thank my wonderful friends and sister for always having my back and making my journey every bit memorable.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF ABBREVIATIONS

LAB	-	Lightness, Color components a and b
RGB	-	Red, Green, Blue
UI	-	User Interface
API	-	Application Programming Interface

1. INTRODUCTION

The use of shredders to cut sensitive documents into pieces has a very interesting history dating back to 1909 when Abbot Augustus Low patented a basic prototype of a shredder. Although intuitive, his patent was never manufactured on an industrial scale until 1935 when a German inventor used a pasta maker to shred some of his sensitive ideological fliers. The principles of a shredder are essentially the same as a pasta maker and a hand crank shredder was more than sufficient to shred a few papers.

The appeal of shredders was restricted to government agencies until the 1980s when civilians started using shredders amid growing awareness about privacy and concerns about the breach of it. Standard shredders cut a document into thin strips of specified width. During the infamous U.S embassy takeover in Iran, many of the shredded documents belonging to American interests were successfully de-shredded. This led to the Defense department’s interest in de-shredding algorithms to explore vulnerabilities in existing system to develop more sophisticated shredders.

The principles of de-shredding are like jigsaw puzzle solvers. The guiding principle is all about identifying the neighboring piece for each selected piece and the final output is expected to reproduce the original image. The similarity ends there. There are additional challenges associated with shredded documents. In most of the existing literature, the jigsaw problem focuses on color gradient information along the edges to

calculate the neighboring piece. This system works when there is a lot of contrast color information to work on. Most of the existing algorithms produce best results when the image has clear divisions of regions delineated with contrasting colors. However, a document is predominantly text with almost uniform contrast in grayscale. Hence, the information extracted from a particular piece will almost always look similar to the information extracted from all the other pieces making it difficult to identify the best match.

Some of the existing literature relies on natural language processing to identify an alphabet along the connecting edge and matching it with prediction of the alphabet on every other piece's complementary edge [1]. For instance, if the page is shredded cutting the alphabet 'a', a prediction on the alphabet is made on the right edge and matched with the prediction on the left edge of the other strips. If a strip identifies its left edge alphabet as 'a', then it is identified as the correct neighbor. The biggest problem with this approach is with scalability and limited utility. The main use of de-shredding technology is in defense department. Most of the confiscated documents from enemy lines are expected to be written in non-English languages. This would drastically limit the utility of the above-mentioned model.

Another proposed solution involves classifying the strips into categories and using existing algorithms for piece assembly [2]. The focus of the algorithms is on identifying the first piece to be used for placement. This piece is also called seed piece [3] which determines the efficacy of the algorithm in solving a given puzzle. This solution,

although much better than using existing jigsaw algorithms, suffers from the problem of identifying appropriate metrics for matching piece identification. This approach would work best for placement of pieces once appropriate metrics are identified.

The proposed algorithm for de-shredding documents takes into consideration the nature of the images which is grayscale documents and not in the LAB color space with contrast information. A new metric for determining the best buddy shred has been proposed and it is used as the core metric around which the shreds are reassembled.

Chapter 2 gives information about the current metrics being used for best buddy piece identification. Chapter 3 discusses the setup that was done for this project including the methodologies adopted for setting up the android application. Chapter 4 explains the design constraints and how they were overcome in this project. Chapter 5 presents the core implementation details of the project. Chapter 6 shows some of the test runs and accuracy graphs for various piece counts. Finally, Chapter 7 has the conclusion.

2. BACKGROUND

Most of the de-shredding tasks currently being performed require the use of some amount of manual intervention. They do not completely automate the process of de-shredding. The inspiration for fully automated de-shredder comes from the use of fully automated algorithms for jigsaw puzzle solvers. Two of the most relevant works which form the core of the proposed de-shredder will be discussed here before moving on to existing work.

The concept of using a seed piece around which all other pieces in a jigsaw puzzle are placed was pioneered by Cho et al. in 2010. The concept of best buddy neighbor was first introduced by Cho et al. This means that when two pieces Px and Py were to be matched on their complementary sides Sx and Sy, and they were determined to be best buddies, there is no other piece Pz with better compatibility on Sz with Sx than Sy on Py.

This idea that was introduced by Cho et al. was further expanded by Tal and Paikin whose work is considered as pioneering in the field of mixed bag puzzles [4]. This is the case where the number of total puzzles are known to be greater than one and there are many missing pieces too. This approach relies on a kernel growth approach where a threshold score is maintained and once the threshold falls below a certain value, a new

puzzle is created. This approach is going to form the basis of the work to be done in de-shredding documents. Even though this approach is suitable for images, the best buddy metric used in Tal and Paikin solver does not work well on documents.

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
```

Figure 1. Tal Paikin Placer Algorithm

The proposed pixel correlation metric algorithm calculates pixel correlation between any two complementary edges of two shreds instead of predicting adjacent edge based on color gradient change. This approach is more effective for processing grayscale documents because RGB information does not give sufficient information for predicting adjacent edges when the contrast information remains the same.

3. DEVELOPING THE APPLICATION

The first step to building an Android app is to learn the Android architecture and Android model of message passing. The following terms are necessary to understand the basic architecture of our 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 goal was to capture image using the camera and pass the “Intent” to a new activity to handle the de-shredding operation. The structural elements of the final android app are established through this process. Figure 2 shows the user interface (UI) of the app.

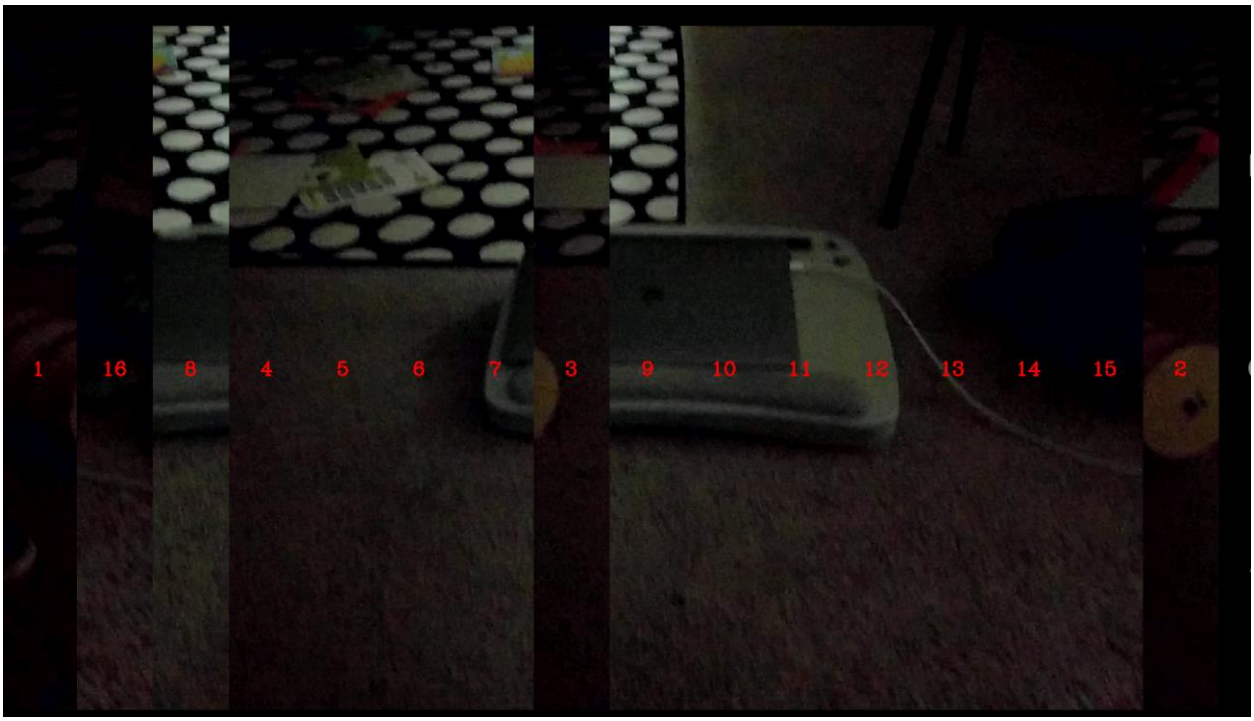

Figure 2. App interface for de-shredding documents

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 3.

```
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
```

Figure 3. 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;
    }
}
```

Figure 4. Communication between ImageCaptureActivity and SplitImageActivity

The Android manifest file had to be modified to grant permissions for camera and external storage because the final assembled image is stored in the internal or SD card storage portion of the Android phone.

In addition to modifying the Android manifest file, it is also important to modify the permissions within the application screen of the app. Figure 4 displays exactly how it is done after going into App info section.

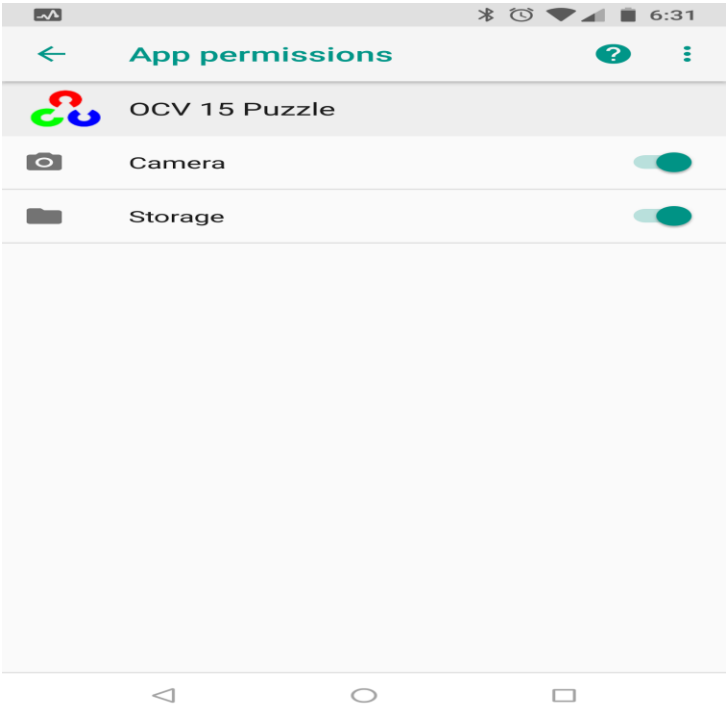

Figure 5. Granting permissions to the camera and storage

The proposed de-shredding algorithm within our app has been implemented primarily using openCV library for Android. This requires the download of a specific application called “openCV manager” into the Android phone prior to performing any openCV

related activity. It is also important to download the openCV library into the Android studio during the creation of the project.

OpenCV libraries are extensively used by our app in extracting the pixel values from each shred to perform comparison operations as well as to calculate the metrics on which the best neighboring buddy is calculated. The steps to include openCV library in Android studio are explained below:

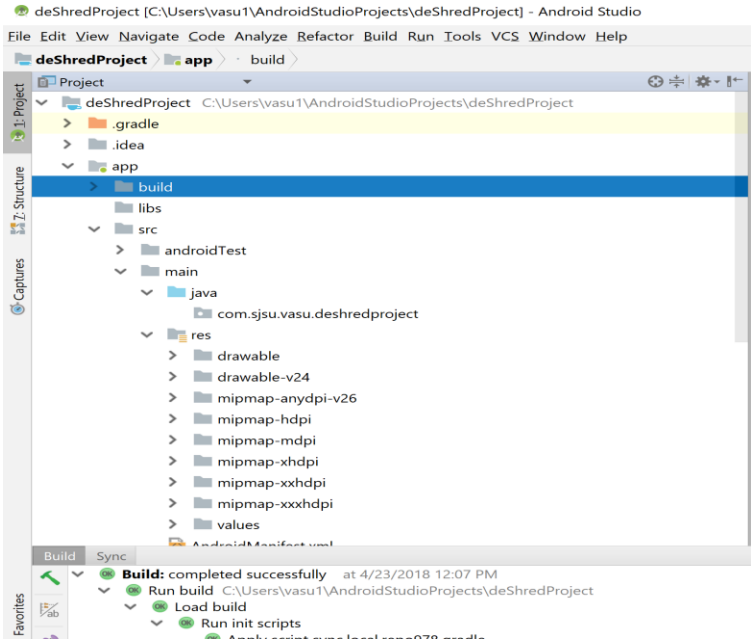

Figure 6. Structure of the project

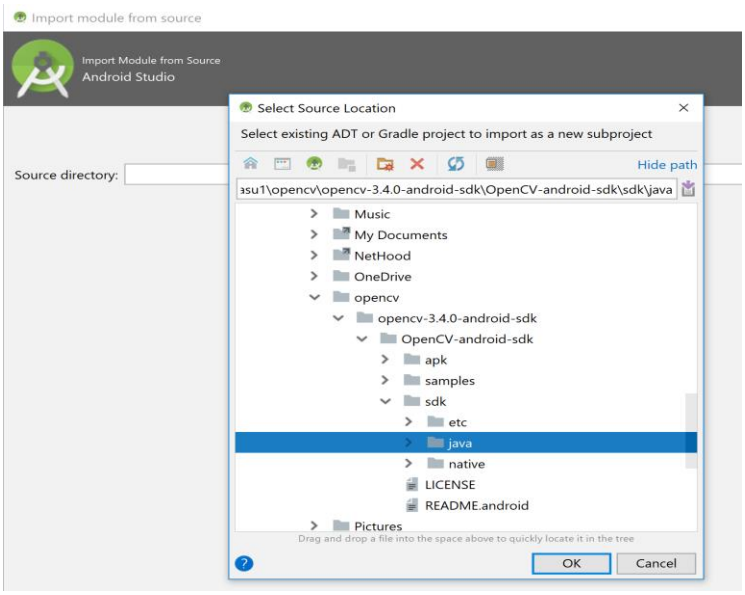

Figure 7. Import openCV module into Android studio

4. DESIGN

Segmentation of shreds into portions

Assembling shreds into whole pages involves identifying how the initial shreds are going to be processed. In order to aid in that, an approach needs to be framed to determine how to extract individual shreds. There are two possible approaches: In one of the approaches, a whole document could be photographed and programmatically divided into shreds and jumbled before performing an assembly. This approach does not give the user the freedom to arrange the shreds according to their wish and instead performs the jumble programmatically.

The other approach could be to place the shreds on a neutral background with constant spacing between the individual shreds and taking photograph of the arranged shreds. The application determines the boundary of the shred and collects the pieces into independent Matrix objects. One advantage of this approach is that it gives sufficient freedom to the user for arranging the shreds instead of taking control programmatically. However, it is crucial that the shreds are identified and separated from the neutral backgrounds to accurately get the background pixel information.

In this project, the first approach was used to test the accuracy of the proposed metric for determining best possible matching shred. The second approach was tested after making sure that the existing metric works on documents with words instead of images.

De-shredding documents versus RGB images

The difference between the problem statement taken up in this project and the prior work is that, the prior work was focussed on images with red, green and blue color components to them. These are images with RGB pixels whose color gradient changes are calculated, and the color gradient change is used to predict the next set of pixels and the predicted values are compared to actual pixel values along the complementary border edge of actual shred pieces and the piece with most similarity to the predicted gradient value is chosen as the best neighbor piece.

However, this application is mainly aimed at assembling de-shredded documents which have uniform pixel distribution and hence the existing metrics will not yield the required results. The weak spot with respect to the existing algorithm for re-assembling puzzle pieces is in the mechanism for determining best neighbors. Figure 7 shows the run of the existing algorithm developed by Hammoudeh and Pollett on a document image. The prior work was tested extensively on RGB images [3] but this experiment was conducted in this project to test on grayscale document images.

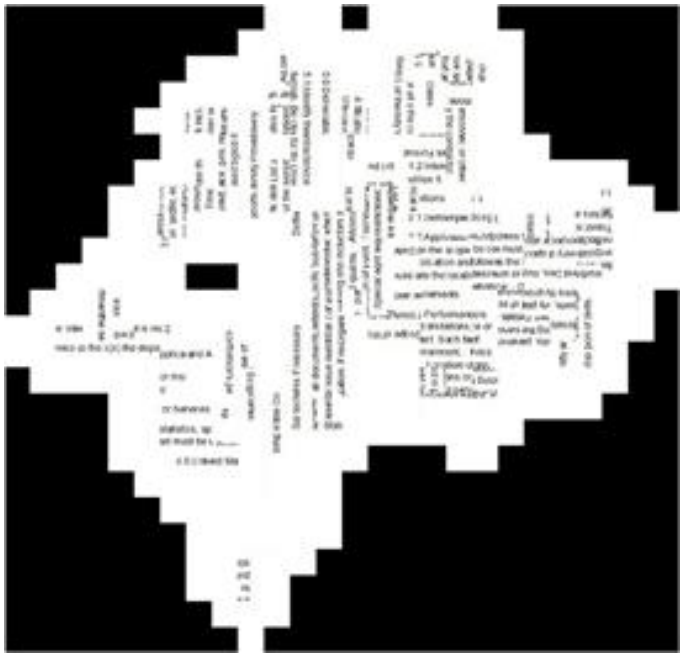

Figure 8. Efficacy of existing algorithms on document image

The hierarchical clustering of the document images is given by the following Fig 8.

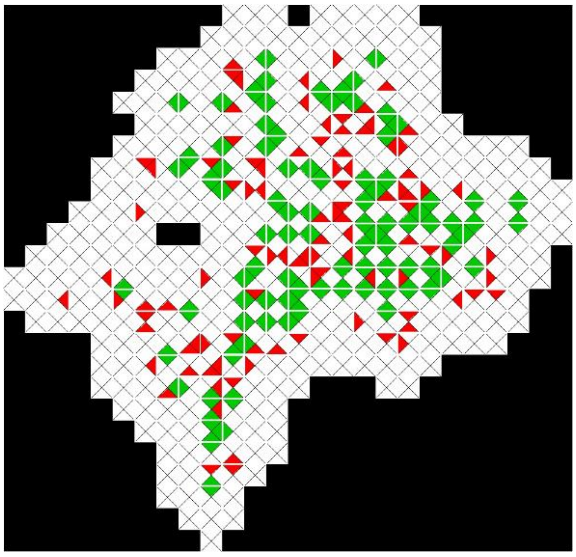

Figure 9. Weak hierarchical clustering of document images

Computational cost of adjacency matrix calculation

In the existing work on jigsaw puzzle solver, each piece has four sides s1, s2, s3 and s4 and each side of piece Pi is matched with all four sides of piece Pj. This would mean that the adjacency matrix for each piece Pi is an enormous three dimensional matrix of the dimensions Adj[total_nunber_of_pieces-1][number_of_sides][number_of_sides] which is stored for each piece. A puzzle board with sixteen pieces will have sixteen adjacency matrices with the dimension of [15][4][4]. The extent of computation intensity of this operation is given by the snapshot shown in Fig. 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);
```

Figure 10. Computational complexity in practice

The computational intensity of these operations can be mitigated by parallel processing of adjacency matrix calculations. This might not help with the speedup on a significant level. The proposed document de-shredder algorithm addresses this issue by assigning either 1s or 0s on the matrix corresponding to when a pixel value is either present or absent. It is illustrated as shown below.

[Matching degree11 Matching degree12 Matching degree13]

[Matching degree21 Matching degree22 Matching degree23]

[Matching degree31 Matching degree32 Matching degree33]

.

.

.[Matching degree n1. . . . . Matching degree nn]

where,

Matching degree n_i = similarity metric of left side of piece n with right side of piece i

5. IMPLEMENTATION

The implementation portions can be divided into the three aspects as shown in the following figure.

Figure 11. Implementation focus areas

Algorithm

The algorithm that is used in this project is based on the Paikin and Tal solver for jigsaw puzzles. The Tal and Paikin algorithm used the following metrics for calculating the best neighboring piece and it can be seen below.

$$\left(2 * P_{ik} - P_{ik-1} - P_{j-1}\right) + (P_{j1} + P_{j2} - P_{ik})$$

Due to the issues mentioned in the prior sections of the document, the proposed metric we use is:

Matching degree$_{ij}$= (a+b)/(a+b+c+d)

where,

a = AND operation on corresponding pixels on piece i and piece j

b = AND operation on the complementary pixels on piece i and piece j

c = AND operation between pixel on piece i with the corresponding pixel's complementary value on piece j

d = AND operation between pixel on piece j with corresponding pixel's complementary value on piece i

For example,

If rightmost edge of piece Pi is, [0,0,1,1,1,0,1,1],

Leftmost edge of piece Pj is, [1,1,0,0,1,1,0,1]

Matching degree$_{ij}$ = (a+b)/(a+b+c+d)

a = (0.1)+(0.1)+(1.0)+(1.0)+(1.1)+(0.1)+(1.0)+(1.1)

a = 1+1 = 2

b = (1.0)+(1.0)+(0.1)+(0.1)+(0.0)+(1.0)+(0.1)+(0.0)

b = 0

c = (0.0)+(0.0)+(1.1)+(1.1)+(1.0)+(0.0)+(1.1)+(1.0)

c = 1+1+1 = 3

d = (1.1)+(1.1)+(0.0)+(0.0)+(0.1)+(1.1)+(0.0)+(0.1)

d = 1+1+1 = 3

Matching $degree_{ij}$ = (2+0)/(2+0+3+3) = 2/8 = ¼ = 0.25

Events API

The Android platform for the application is entirely event-driven. This means that, from the time the photographs of the shreds is taken to the time the re-assembled output is delivered, the operations to be performed are based on events that are performed by the user. The APIs used in this project fall under the Camera section of Android APIs. The following are some of the main APIs used in this project.

onCreate(): This API helps initialize the activity screen with a view which is a window where most of the UI operations are performed. Hence, it is very common to find the setContentView() method inside the definition of this API.

onCameraViewStarted(int width, int height): This API is used to begin processing the camera frames. In this project, this API is used to initialize the puzzle board size and then the onCameraFrame API is called to begin processing the image frames from the camera. It is imperative to set the board size in this method because it is what determined what needs to be captured and sent for further processing.

onTouch(View view, MotionEvent event): This API determined what needs to be done if any area covered by the previously determined puzzle board is touched. In the

case of this project, this is used to determine if the touch event is performed inside the game board. If so, the de-shredding algorithm is performed. If the touch event happens outside the puzzle game board, no action is taken.

onCameraFrame(Mat inputFrame): This method is used to take in the image frames that are sent from the camera to the view. In the case of this project, the input image is taken and divided into sub-regions using the sub-mat function of the matrix Mat object and the sub-mats are jumbled in a random order to simulate the concept of shreds. In this method, the re-assembled shreds are stored into the disk by getting converted to bitmap format. The code snippet in the following figure illustrates this.

```
public Mat onCameraFrame(Mat inputFrame) {
    if (numImagesUpdated > 0) {
        if (numImagesUpdated == 1) {
            lastFrame = inputFrame.clone();
        }
        --numImagesUpdated;
        return mPuzzle15.puzzleFrame(inputFrame);
    }

    Mat finalOutputFrame = mPuzzle15.puzzleFrame(lastFrame);
    Mat finalSavedFrame = mPuzzle15.savedFrame(lastFrame);
    //Process the frame and de-shred the pieces and store the final image on the disk
    Bitmap bmp = null;
    try
    {
        bmp = Bitmap.createBitmap(finalSavedFrame.cols(), finalSavedFrame.rows(), Bitmap.Config.ARGB_8888);
        Utils.matToBitmap(finalSavedFrame, bmp);
    }
    catch (Exception e){Log.d( tag: "Exception",e.getMessage());}

    String savedLocation = saveToInternalStorage(bmp);
    Log.d( tag: "savedloc",savedLocation);
```

Figure 12. Code snippet to illustrate onCameraFrame method usage

Storage and Permissions

This project involves storing the results onto the external storage device of Android. This involves external storage permissions to be granted to the application. The AndroidManifest.xml file is used to explicitly add permissions to the storage device. The following figure illustrates how the manifest file looks when external storage permissions are requested.

```
<application
    android:icon="@drawable/icon"
    android:label="OCV 15 Puzzle"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen" >

    <activity
        android:name=".Puzzle15Activity"
        android:label="OCV 15 Puzzle"
        android:screenOrientation="landscape"
        android:configChanges="keyboardHidden|orientation" >

        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<uses-feature android:name="android.hardware.camera.front" android:required="false"/>
<uses-feature android:name="android.hardware.camera.front.autofocus" android:required="false"/>
```

Figure 13. Manifest file with external storage permissions added to it

6. RESULTS

The initial algorithm was run with the metrics for RGB colors and it produced the following result:

```
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]
```

Figure 14. With color images varied distances

```
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
Finished Asymmetric Compatibility
Identified Best Buddy Candidates for 100 pieces
Finished Mutual Compatibility
Best buddy candidates for piece 0 is [0, 1, 1, 0]
```

Figure 15. With black and white images the distances are almost the same

It can be seen that when the initial algorithm is run with black and white images, it gave results very similar to each other in terms of distance and it resulted in a tie situation with the same piece vying for the minimum distance position. This leads to inaccuracy.

However, running with the modified algorithm resulted in better accuracy and the following figure shows a shredded document being assembled.

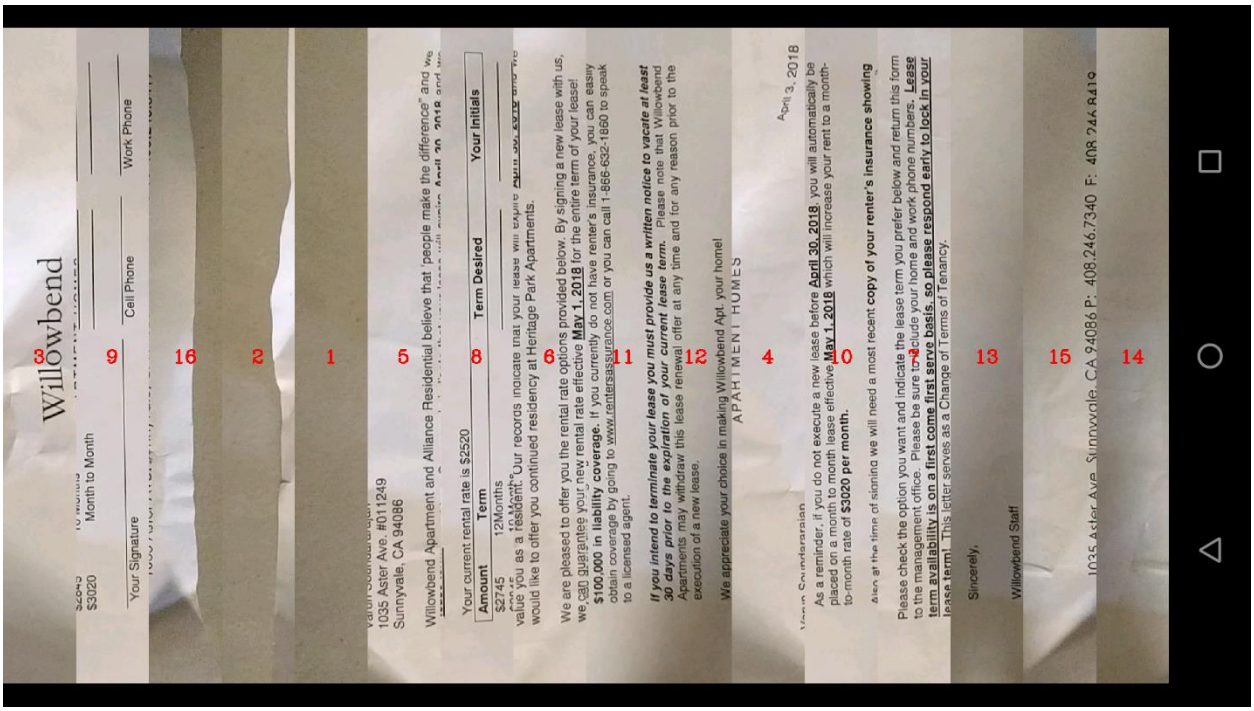

Figure 16. Jumbled shreds

The shreds are jumbled up programmatically when the picture of whole document is taken as shown above. The shreds have been numbered to figure identify errors if there is any.

Figure 17 shows the re-assembled shreds when any region within the camera frame is clicked. Ideally, when there are zero errors, the shreds will be arranged sequentially after final assembly.

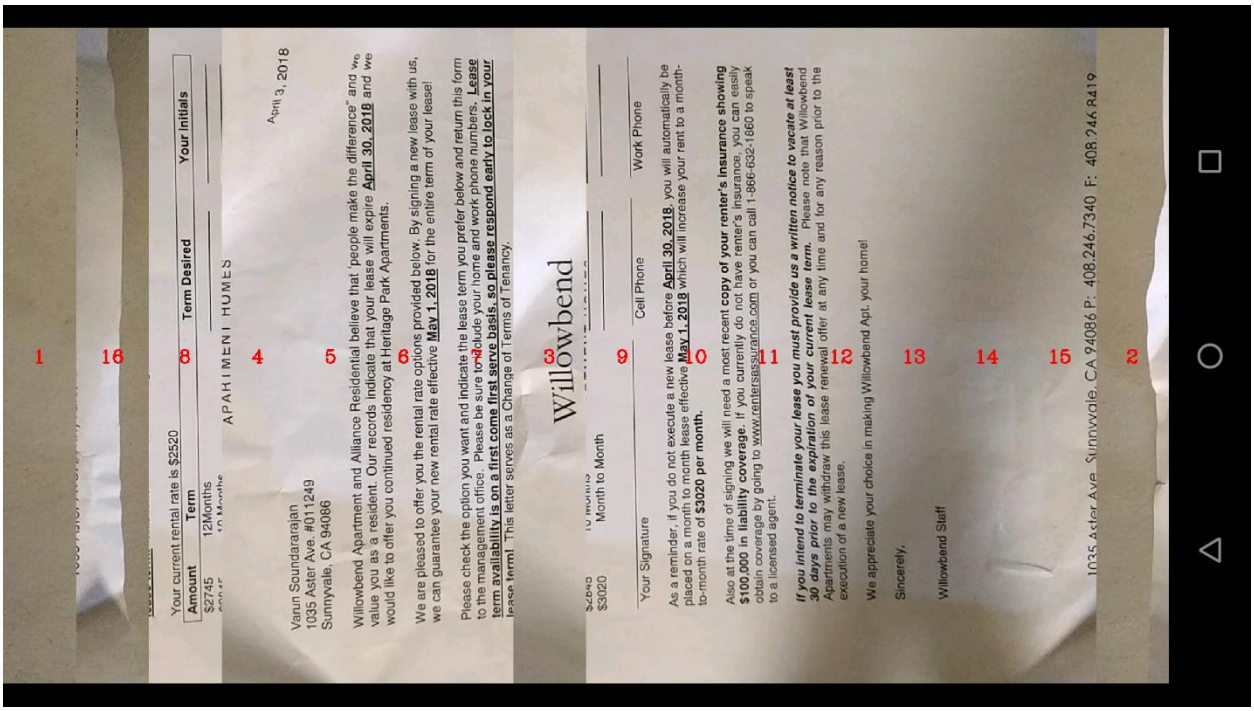

Figure 17. Rearranged shreds

The following Figure 18 shows the plot of values between number of shreds and the number of errors. It can be seen that the number of errors have stayed at a reasonable value and even though the number of errors increase with increase in number of shreds, the error difference has stayed pretty close.

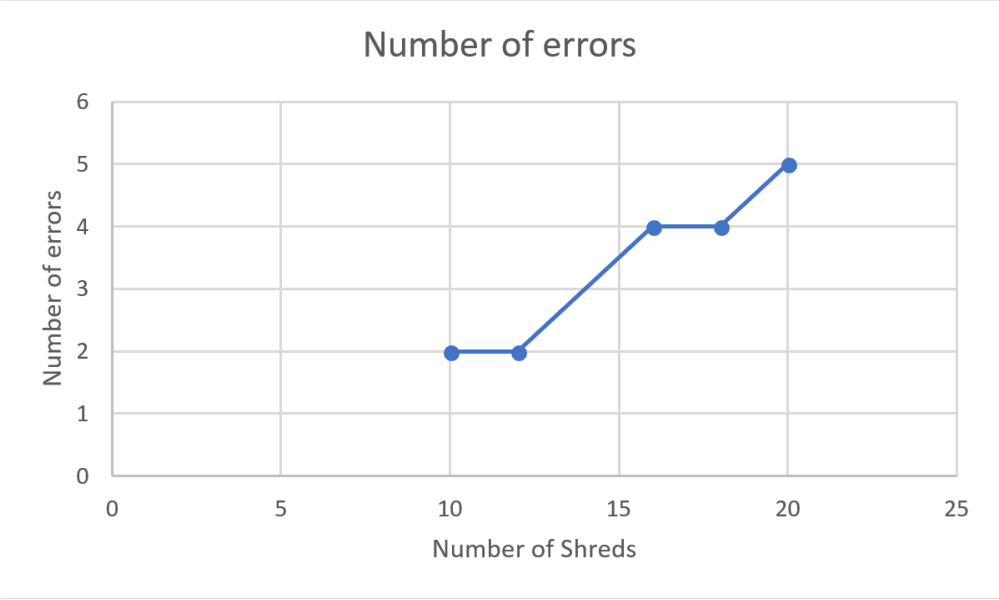

Figure 18. Number of shreds vs. Number of errors

Some experiments were also conducted where the current algorithm is compared with the prior work for document images and the following chart illustrates the difference in performance. The performance difference is very huge because the currently existing solution is tailored to RGB images.

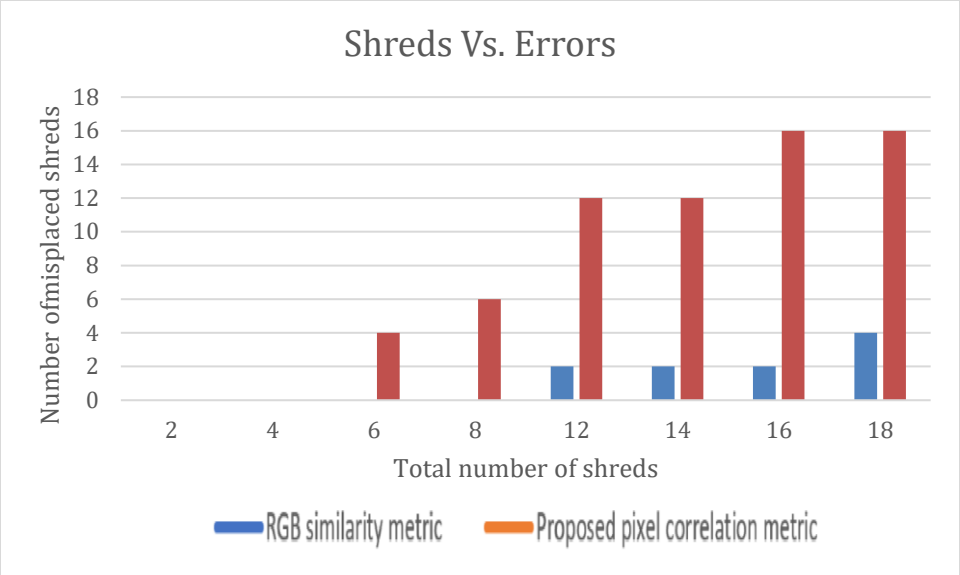

Figure 19. Comparing RGB similarity and pixel correlation metric

7. CONCLUSION

This project focused on implementing an app that would assemble de-shredded documents into whole documents with reasonable accuracy. This problem is similar in nature to the problem of assembling puzzle pieces in a jigsaw. Hence, existing solutions for jigsaw puzzle assembly were used as base models upon which this solution was built. De-shredding varies in many aspects from the jigsaw puzzle solvers. So, it was imperative that some fundamental changes had to be made on the existing models which led to the proposal of new metric for best neighboring piece match.

Jigsaw puzzle solvers worked by selecting a seed piece which was initially placed on the puzzle board and corresponding best neighbors were identified for each of the four

sides on the seed piece. In turn, each of those neighbors identified their best neighbors on each of their sides resulting in a cluster of sixteen pieces. This pattern was repeated with a different seed piece and its corresponding sixteen neighbors. Several such clusters were created and stitched using hierarchical clustering.

However, the above method had a key problem of identifying the best metric for picking the best possible neighboring piece. The existing metrics were heavily reliant on color schemes and color gradient change. This was not working on document images. Hence, a new metric was proposed in this project where presence or absence of pixels along the border was determined and it was used as a metric for the identification of neighboring pixels.

This algorithm was incorporated into an Android app which would take as input a picture of the shreds or take a whole document and programmatically shred it to pieces, process the shreds using openCV library and assemble the shreds into a bitmap image. This bitmap image is then internally stored into the Android storage. Permissions for accessing the storage are obtained using the Android manifest file.

Several experiments were conducted to compare the prior work RGB metric algorithm with the pixel correlation metric algorithm proposed in this project. It was seen that the proposed algorithm performed with consistent accuracy rate of 90% and above for grayscale document images using the modified correlation metrics. However, the existing RGB metric algorithm performed with accuracy of only 10% with grayscale

documents. It returned several tied shreds as candidates for neighbors. The performance of both the RGB metric and the proposed pixel correlation metric remained almost the same at 90% and the new algorithm did not show any significant improvement on high contrast images with rich RGB value variations.

There are various techniques to identify the best possible metrics. We explored many such methods including curve fitting, natural language processing. The problem with natural language processing methods was that they restricted the language of the document which makes this solution non-generic. Moreover, the primary use of de-shredding is in the defense department which requires the ability to process shreds in several foreign languages.

Curve fitting algorithms made the solution more complicated for a fast Android application. Hence, a simpler and more effective metric was utilized for the purpose of fast processing of shredded strips.

REFERENCES

[1] Z. Hammoudeh and C. Pollett, "Clustering-Based, Fully Automated Mixed-Bag Jigsaw Puzzle Solving," in *Proceedings of the Conference on Computer Analysis of Images and Patterns*, 2017.

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

[3] F.-C. W. Lin HY., "Image-Based Techniques for Shredded Document Reconstruction.," in *Advances in Image and Video Technology. PSIVT*, Berlin, 2009.

[4] H. Q. a. J. L. Y. Liu, "Shredded Document Reconstruction Based on Intelligent Algorithms," in *International Conference on Computational Science and Computational Intelligence (CSCI)*, 2014.

[5] M. D. F. K. J. Pearl, "Strip shredded document reconstruction using optical character recognition," in *International Conference on Imaging for Crime Detection and Prevention*, 2011.

[6] Z. Z. D. H. L. Zhu, "Globally Consistent Reconstruction of Ripped-Up Documents," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 30, no. 1, 2008.