THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FROM IMAGE TO WORDS: COMPUTER VISION AND MACHINE LEARNING

ALISHA REGE
SPRING 2016

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Engineering
with honors in Computer Science

Reviewed and approved* by the following:

Robert Collins
Associate Professor of Computer Science and Engineering
Thesis Supervisor

John Hannan
Associate Professor of Computer Science and Engineering
Interim Associate Department Head
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

ABSTRACT

Approximately 8 million people from ages 4 to 75 are visually disabled. [1] These people have difficulty integrating into society because of their inability to see the world around them. Many visually disabled individuals find it hard to navigate around an unfamiliar town by themselves as there is no clear visually disabled path. This can make travelling hard when they go on business or pleasure related trips. This need to communicate with the visual world begs for a solution. To this end, I decided to develop an app that would read written text aloud. Using computer vision and machine learning techniques, this program will be able to take any image as input and output a sound that corresponds with the words seen. This program was able to read 40% of the text seen out loud correctly. With this technology, an application for any mobile device can be made to take pictures of the surroundings and read any visually discernable text audibly.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Special thanks to Dr. Robert Collins for supervising my thesis and to Dr. John Hannan for reading over the thesis.

Chapter 1

Introduction

The world is full of infinite senses. Each sense brings us new information and new knowledge. However, for people with disabilities, learning new knowledge is difficult without access. This makes it hard for them to grasp information that cannot be sensed by their bodies. One of these disabilities is the inability to read. A blind individual cannot read signs/books that do not exist in audio form unless they have another person read it out loud to them. This disability also disallows them from taking visual cues from the environment around them. This need for a visual aid prompts a need for a solution.

The goal of this project is to cater to the need of visually disabled individuals. This goal is very broad, so the first step towards the answer is to develop a program that converts images to sound. This program could be used by blind people to take in the visual cues around them. This would allow them to travel and read on their own time. This would also reduce the social liability of not catering to the blind.

Previous work by University of California details how text can be found in natural scenes. [2] This work performed statistical analysis of a labeled true data set and determined which features are indicative of text and have low entropy. Then these classifiers were put into an AdaBoost machine learning algorithm to create a better training model. This was then used to test a set of images. This work presents the basis for my work. After the photos are taken and the text based area is discovered, my program can be used to read the text.

To test the datasets, I chose to use images of pages from the book *Pride and Prejudice*. In order to avoid variables such as light variation and shadowing, I used the files from the ebook

version of the text as opposed to the print version. Although the book was electronically published, a study by University of Maryland states that there is little difference between an image on the computer and an image taken by a camera. [3]

As there are many people with visual disabilities, this work is meant to be an aid to those that want to be able to interact with the real world. Although this project works only on a laptop, it is written in Java so that people can implement it on their phone. Combined with other technologies such as sign detection, this project package can help translate signs so disabled individuals can take a walk and be able to read caution signs. This project uses machine learning and computer vision techniques to achieve its goal. This thesis covers the basics of the project from extracting a database on which to train, to training a database using Sift, KMeans, and a Support Vector Machine for different characters to testing images from the book.

Chapter 2

The Training Set: Extracting Images of True Font Files

A computer is similar to the human brain; in order for it to understand what any object/image is, it has to have precedent. So, to make a computer learn how to differentiate letters, it has to learn what every letter looks like; this can be done with a supervised machine learning program. As there is no character image database to train on, one had to be created. To hasten the process, I developed a script to extract images of the training set from a true font file and save each character to a directory. I chose to write this in Java in order to increase efficiency. JavaScript contains a library called OpenType which allows for the manipulation of true font files. Using OpenType, the project was able to trace the identifiable points of the letters and save them as an image. This was partially due to the work of Steve Hanov. [1] His work introduced the library OpenType and gave examples of how to use it to manipulate the true font file.

The script opens up a webpage that truefont files can be dropped into. This then outputs a series of character images. This is shown in Figure 1.

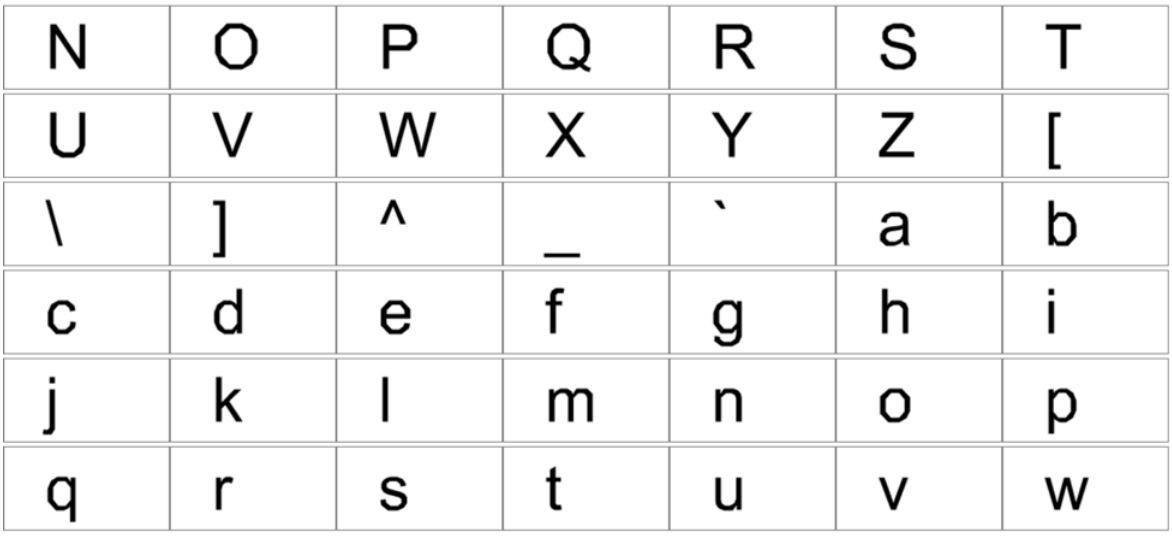

Figure 1: Letters from True Font File

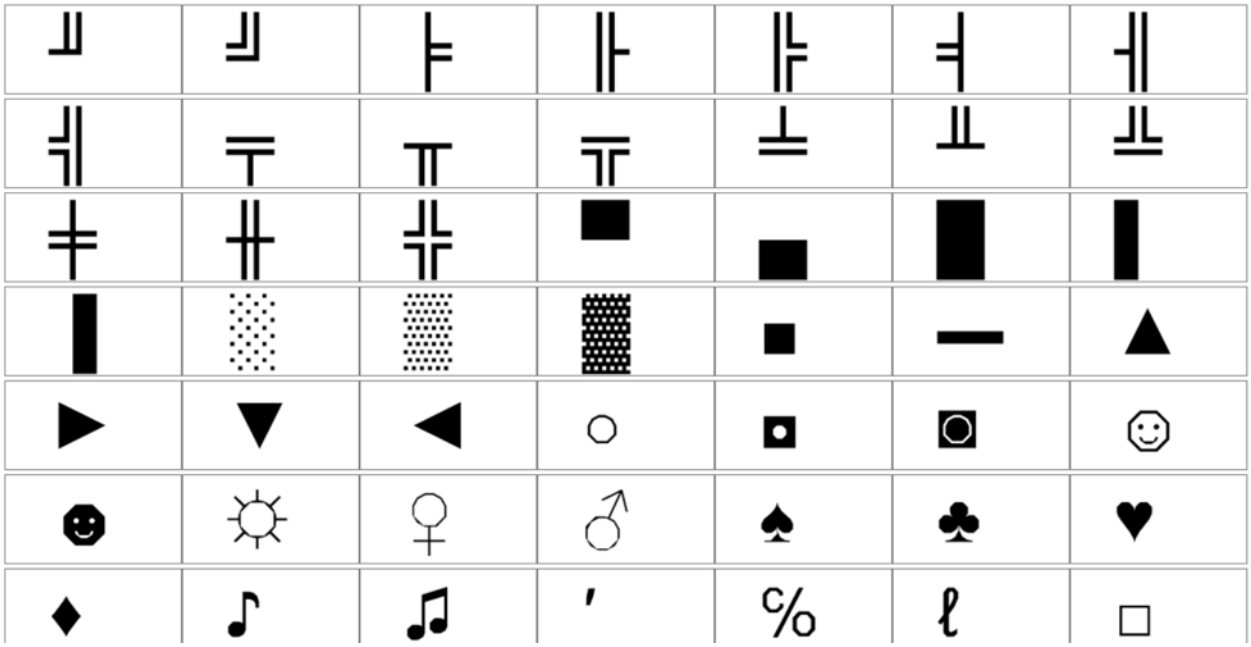

Figure 2: Not Useful Characters

However, the program also outputs characters such as Figure 2 which are not useful. Hence the script was constrained to just use characters 4 – 93 by using the lines: (can be found in Appendix B as getFontPics.html)

```
if (i > 3 && i < 94) {
        var name = (i.toString()).concat(".png");
                canvas.toBlob(function(blob){
                saveAs(blob, name);
        });
}
```

There is potential in further projects in applying these methods to various mathematical symbols as well. Because of Chrome's settings, HTML files have a hard time placing images in specific directories, a Python script was used to separate them into their directories. This can be found listed as separate.py in Appendix B.

Chapter 3

Training the Data Set

Introduction

With the characters in hand and using the Bag of Words training model, I developed a training algorithm for use with the program. This model operates by extracting features from images and creating a "visual vocabulary". This vocabulary is built up of how features correlate to a certain object type. This is done by quantizing the features and representing the images by the frequencies of the features.

Each image broken down in this project used a Bag of Words training model for the training of the data. First each picture was read in and converted into a binary image. The threshold for the black and white conversion of the images was not greatly pertinent due to the fact that the images were created by drawing. This caused a distinct difference in a black value and white value. These images were then placed into a multidimensional array where the length of the array was the number of characters (90) and the width of the array was the number of training sets (4). Only four fonts were used for classification because each image contributed upwards of a 100 feature points and it was not necessary to have more true models. Furthermore, since there are many different fonts ranging from Serif to Non Serif to Cursive, I chose to use only Serif fonts in the training of the data. This included Times New Roman, Georgia and Courier New.

SIFT

Scale Invariant Feature Transform (SIFT) is a technique used to extract identifying features on an image. These features are usually corners. Unlike many other feature detectors, SIFT functions by finding the corners in an image regardless of the scale of the image. These features are then returned as feature

points. The program devised breaks down a feature point into a 4x8 grid. Each of these boxes, then has a feature vector associated with it. These feature vectors go on to form 1x32 arrays that hold descriptor vectors comprised of histograms made of gradient orientations, weighted by magnitude. The formulas in Figure 3 explain the formulas used to compute the magnitude and orientation of the feature.

$$m(x,y) = \sqrt{\left(L(x+1,y) - L(x-1,y)\right)^2 + \left(L(x,y+1) - L(x,y-1)\right)^2}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{\left(L(x,y+1) - L(x,y-1)\right)}{\left(L(x+1,y) - L(x-1,y)\right)}\right)$$

Figure 3: Formulas for Magnitude and Orientation

These descriptors serve as 32 dimensional points for the next part of the program. For example in the letter A shown in Figure 4, the feature locations are circled. This code was taken from CMU's 10-615 Art

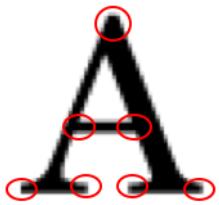

Figure 4: Letter A Features

That Learns class [2]. The feature descriptor vectors are passed to KMeans.

KMeans

This algorithm serves to create “bags” for each descriptor to identify as a part of. Using KMeans, the 32 dimensional features are clustered together into different groups. Since 3775 features were found in all the training images, the features were grouped into 50 clusters. This works by assigning a group of features to one cluster and iteratively updating the groups so that each group has a roughly equal number of features. To illustrate how this works, figure 4 shows a hypothetical scenario in 2 dimensional space with 5 clusters. The Red Dots illustrate how the A features would theoretically be placed.

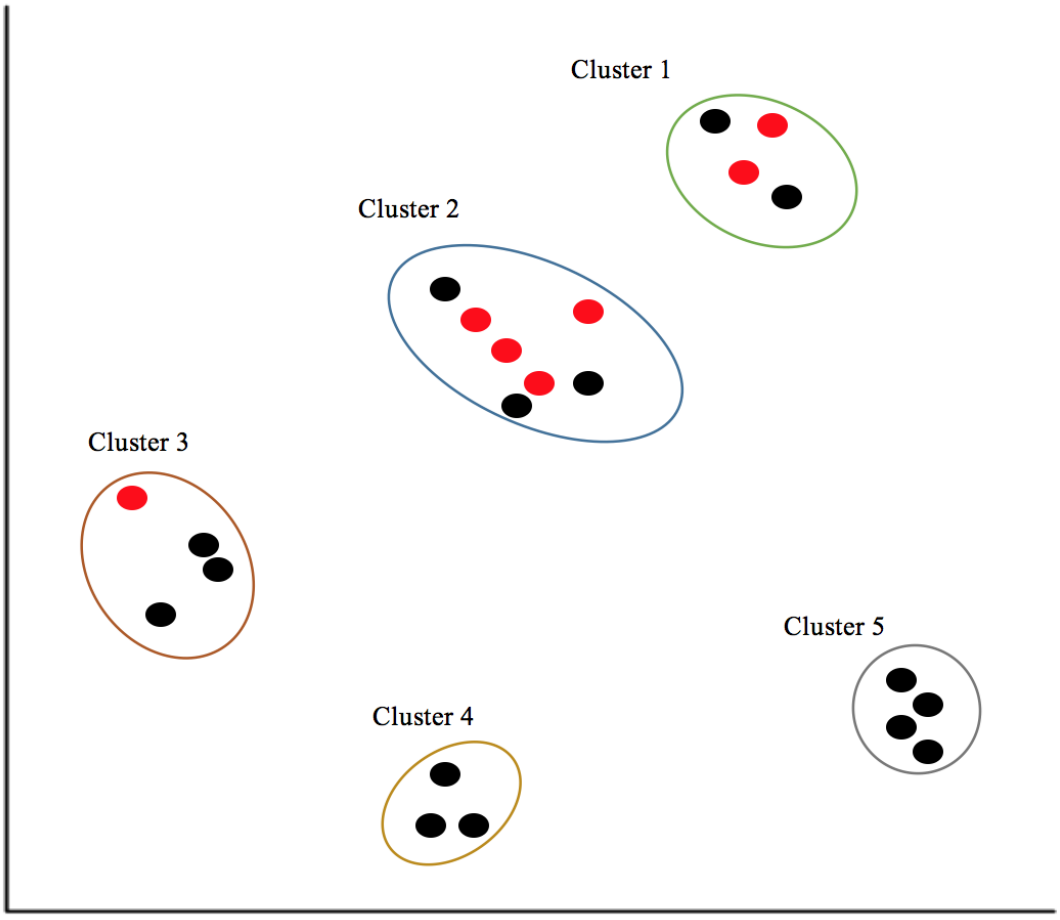

Figure 5: Graph of Features

In this example, the four bottom features of A are in Cluster 2, the two middle features in Cluster 1, and the top feature in Cluster 3. Clusters 1-5 along with the labels of each point are then sent to the SVM to be trained. This code was developed by using the KMeans library in OpenCV library. [3]

SVM

A Support Vector Machine is a useful machine learning technique to train an algorithm how to classify an image. A categorical SVM was used in this case to teach the computer what features are associated with different letters. The features taken from SIFT were categorized into the separate clusters and a table was made of how many features found in every image were from each cluster. Table 1 shows an example of how features in image A would be sent in.

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
2	4	1	0	0

Table 1: Example of Image Classification Sent to SVM

This table was made for each image and sent in to be trained. A label vector was also sent in that detailed that index 0 belonged to a picture of A and index 1 belonged to a picture of B and so on. The support vector machine created linear boundaries between each category and allowed the system to differentiate between the letters. This was also accomplished by using OpenCV's library. [3]

Chapter 4

Extracting Letters from the Test Image

In order to classify the words on the test image, the image had to be broken down into its individual letters. First, I converted the image to black and white. In order to remove any impurities, I then experimented with the threshold to find the best possible results. Figure 6 depicts an example of how impurities can create inconsistency. The image was inverted to show the noise in the picture. This low threshold could cause many letters to be seen as one.

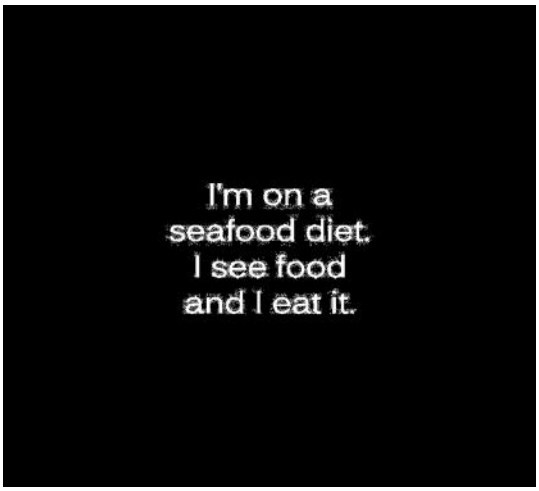

Figure 6: Inverted Impure Image (Low Threshold)

After much trial and error, the threshold was set to 510 for the test pictures. Any value below this threshold was set to black and value above this threshold was set to white. This inverted picture could be seen in Figure 7.

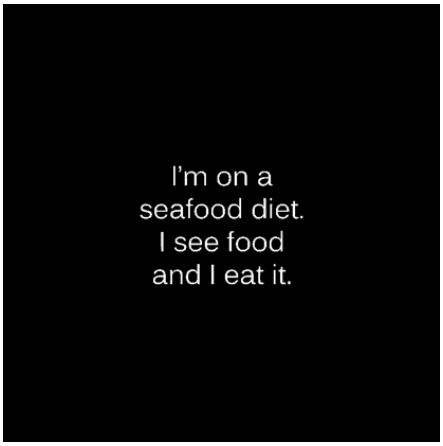

Figure 7: Inverted Pure Image (Proper Threshold)

After setting the threshold, the pictures were broken down into rows. This was done by summing up the values of the pixels in each row. If the the pixel count was greater than 0, the start of a line was declared. Once the row count turned back into 0, the end of a line was noted. This allowed each line to be separated when the columns were summed. This summation resulted in positive values where the letters existed. This is depicted in Figure 8.

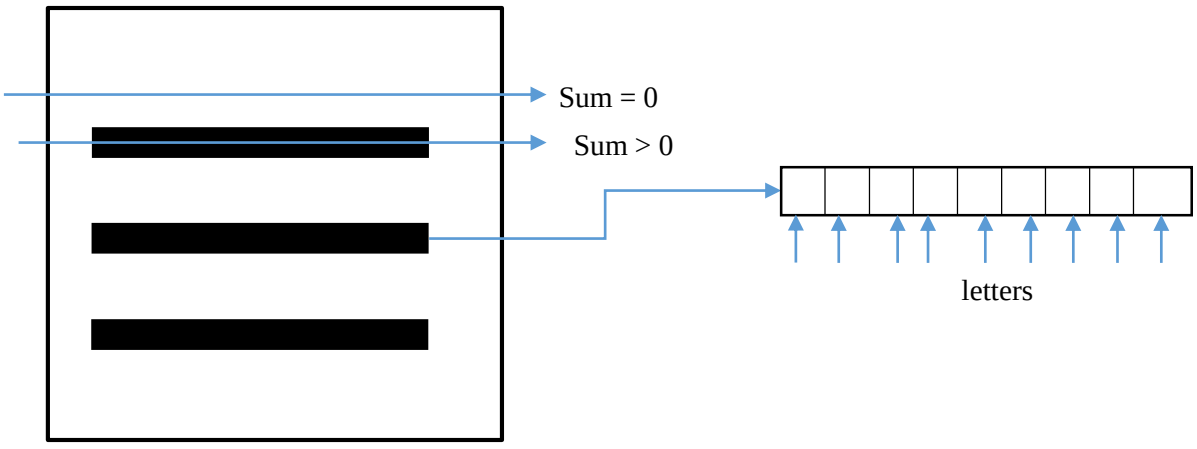

Figure 8: Breaking down Images

One pixel to each side was added and the images were cropped as shown in Figure 9.

Figure 9: Crop Examples

These images were then ready to be used for testing. The images were sent through the same system as the training images. The images were first sent through Sift to find the feature points and then the closest match for the points was found in the center of the clusters created by KMeans in the training phase. These cluster counts were summed up and sent through the predict feature in OpenCV's SVM to find the label that best matched them. Then they were converted to letters using a switch statement.

Chapter 5

Results

This project turned out to have a 30% success rate. The result indicates that bag of words is not a viable approach for reading text. This project was done with no outside light sources and from relatively perfect test images and hence should have had a high correct classification rate. Table 2 shows the break down of the results for the first three pages.

Page 1	48.95%
Page 2	40.15%
Page 3	38.56%

Table 2: Sample of Results

The program was run on 10 pages out of the *Pride and Prejudice* eBook. These results show that as the number of words increased on the page, the classification rate decreased. [4] This is a little alarming. The reason for this error rate could be because the number of categories may have exceeded the number of training images given. Since the library was limited to four fonts, the number of features found in each image may not have been distinguishable enough to classify into 91 categories. A project of this depth may require up to thousands of images for each category. When 10 images of letters were added to the dataset, the results were as follows:

Page 1	49.32%
Page 2	41.23%
Page 3	25.45%

Table 3: Sample of Results with Bigger Training Set

The higher numbers in Table 3 prove that with a larger training set, the numbers are improving logarithmically. For this project to have a success rate above 50 percent, a dataset of more than a thousand images would have to be used.

Chapter 6

Conclusion

Although the project was unsuccessful, it was a good indicator of whether bag of words works for letter classification. Although this specific bag of words model did not work, there are many other alterations which may be possible. The parameters could be played around with in further depth to explore other approaches to the bag of words model, and other models such as neural networks and Bayesian networks could be used. Furthermore, instead of SIFT, feature detection algorithms such as Canny, FAST, or Difference of Gaussians can be used. Furthermore, a deep learning method such as the one used in [5] . This system uses deep learning to teach a computer what letters look like. In future work, this deep learning system can be used for training and the output could be used in java to test the image. This would also reduce the computation time and allow it to be used in an app. Although a low success rate, this project takes the first step towards being able to read text out loud to help integrate the hearing disabled into society.

Appendix A

Code

getFontPics.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8" />
</head>
<body>

<div id=dropTarget style="border:3px dashed #5f90d0; font-
size:30px;width:300px;height:100px;padding:20px;margin:0 auto; text-align: center">Drag TTF file here
</div>
<div id=font-container></div>
<script src="/Users/Alisha/Desktop/Project/jszip.js" type="text/javascript"></script>
<script src="/Users/Alisha/Desktop/Project/canvas-toBlob.js" type="text/javascript"></script>
<script src="/Users/Alisha/Desktop/Project/FileSaver.js" type="text/javascript"></script>
<script src="/Users/Alisha/Desktop/Project/TrueType.js" type="text/javascript"></script>
<script>
$("#dropTarget").on("drop", function(e) {
   setTimeout(function() {layout.go(); }, 500);
});
</script>
<p>
<a href="/Users/Alisha/Desktop/Project/TrueType.js">Source code</a>
<script>
"use strict";

/** @param {string=} message */
function assert(condition, message) {
   message = message || "Assertion failed";
   if (!condition) {
      alert(message);
      throw message;
   }
}


/** @constructor */
function BinaryReader(arrayBuffer)
{
   assert(arrayBuffer instanceof ArrayBuffer);
   this.pos = 0;
   this.data = new Uint8Array(arrayBuffer);
```

```
}

BinaryReader.prototype = {
   seek: function(pos) {
      assert(pos >=0 && pos <= this.data.length);
      var oldPos = this.pos;
      this.pos = pos;
      return oldPos;
   },

   tell: function() {
      return this.pos;
   },

   getUint8: function() {
      assert(this.pos < this.data.length);
      return this.data[this.pos++];
   },

   getUint16: function() {
      return ((this.getUint8() << 8) | this.getUint8()) >>> 0;
   },

   getUint32: function() {
     return this.getInt32() >>> 0;
   },

   getInt16: function() {
      var result = this.getUint16();
      if (result & 0x8000) {
         result -= (1 << 16);
      }
      return result;
   },

   getInt32: function() {
      return ((this.getUint8() << 24) |
            (this.getUint8() << 16) |
            (this.getUint8() <<  8) |
            (this.getUint8()      ));
   },

   getFword: function() {
      return this.getInt16();
   },

   get2Dot14: function() {
      return this.getInt16() / (1 << 14);
   },
```

```
    getFixed: function() {
        return this.getInt32() / (1 << 16);
    },

    getString: function(length) {
        var result = "";
        for(var i = 0; i < length; i++) {
            result += String.fromCharCode(this.getUint8());
        }
        return result;
    },

    getDate: function() {
        var macTime = this.getUint32() * 0x100000000 + this.getUint32();
        var utcTime = macTime * 1000 + Date.UTC(1904, 1, 1);
        return new Date(utcTime);
    }
};

/** @constructor */
function TrueTypeFont(arrayBuffer)
{
    this.file = new BinaryReader(arrayBuffer);
    this.tables = this.readOffsetTables(this.file);
    this.readHeadTable(this.file);
    this.length = this.glyphCount();
}

TrueTypeFont.prototype = {
    readOffsetTables: function(file) {
        var tables = {};
        this.scalarType = file.getUint32();
        var numTables = file.getUint16();
        this.searchRange = file.getUint16();
        this.entrySelector = file.getUint16();
        this.rangeShift = file.getUint16();

        for( var i = 0 ; i < numTables; i++ ) {
            var tag = file.getString(4);
            tables[tag] = {
                checksum: file.getUint32(),
                offset: file.getUint32(),
                length: file.getUint32()
            };

            if (tag !== 'head') {
                assert(this.calculateTableChecksum(file, tables[tag].offset,
                        tables[tag].length) === tables[tag].checksum);
            }
        }
```

```
    return tables;
},

calculateTableChecksum: function(file, offset, length)
{
    var old = file.seek(offset);
    var sum = 0;
    var nlongs = ((length + 3) / 4) | 0;
    while( nlongs-- ) {
        sum = (sum + file.getUint32() & 0xffffffff) >>> 0;
    }

    file.seek(old);
    return sum;
},

readHeadTable: function(file) {
    assert("head" in this.tables);
    file.seek(this.tables["head"].offset);

    this.version = file.getFixed();
    this.fontRevision = file.getFixed();
    this.checksumAdjustment = file.getUint32();
    this.magicNumber = file.getUint32();
    assert(this.magicNumber === 0x5f0f3cf5);
    this.flags = file.getUint16();
    this.unitsPerEm = file.getUint16();
    this.created = file.getDate();
    this.modified = file.getDate();
    this.xMin = file.getFword();
    this.yMin = file.getFword();
    this.xMax = file.getFword();
    this.yMax = file.getFword();
    this.macStyle = file.getUint16();
    this.lowestRecPPEM = file.getUint16();
    this.fontDirectionHint = file.getInt16();
    this.indexToLocFormat = file.getInt16();
    this.glyphDataFormat = file.getInt16();
},

glyphCount: function() {
    assert("maxp" in this.tables);
    var old = this.file.seek(this.tables["maxp"].offset + 4);
    var count = this.file.getUint16();
    this.file.seek(old);
    return count;
},

getGlyphOffset: function(index) {
```

```
    assert("loca" in this.tables);
    var table = this.tables["loca"];
    var file = this.file;
    var offset, old;

    if (this.indexToLocFormat === 1) {
        old = file.seek(table.offset + index * 4);
        offset = file.getUint32();
    } else {
        old = file.seek(table.offset + index * 2);
        offset = file.getUint16() * 2;
    }

    file.seek(old);

    return offset + this.tables["glyf"].offset;
},

readGlyphs: function(file) {
    assert("glyf" in this.tables);
    var glyphTable = this.tables["glyf"];

    file.seek(glyphTable.offset);

    var glyphs = [];

    while(file.tell() < glyphTable.offset + glyphTable.length) {
        glyphs.push(this.readGlyph(file));

        while(file.tell() & 1) {
            file.getUint8();
        }
    }

    return glyphs;
},

readGlyph: function(index) {
    var offset = this.getGlyphOffset(index);
    var file = this.file;

    if (offset >= this.tables["glyf"].offset + this.tables["glyf"].length)
    {
        return null;
    }

    assert(offset >= this.tables["glyf"].offset);
    assert(offset < this.tables["glyf"].offset + this.tables["glyf"].length);

    file.seek(offset);
```

```
    var glyph = {
        numberOfContours: file.getInt16(),
        xMin: file.getFword(),
        yMin: file.getFword(),
        xMax: file.getFword(),
        yMax: file.getFword()
    };

    assert(glyph.numberOfContours >= -1);

    if (glyph.numberOfContours === -1) {
        this.readCompoundGlyph(file, glyph);
    } else {
        this.readSimpleGlyph(file, glyph);
    }

    return glyph;
},

readSimpleGlyph: function(file, glyph) {

    var ON_CURVE       =  1,
        X_IS_BYTE      =  2,
        Y_IS_BYTE      =  4,
        REPEAT         =  8,
        X_DELTA        = 16,
        Y_DELTA        = 32;

    glyph.type = "simple";
    glyph.contourEnds = [];
    var points = glyph.points = [];

    for( var i = 0; i < glyph.numberOfContours; i++ ) {
        glyph.contourEnds.push(file.getUint16());
    }

    // skip over intructions
    file.seek(file.getUint16() + file.tell());

    if (glyph.numberOfContours === 0) {
        return;
    }

    var numPoints = Math.max.apply(null, glyph.contourEnds) + 1;

    var flags = [];

    for( i = 0; i < numPoints; i++ ) {
        var flag = file.getUint8();
```

```
        flags.push(flag);
        points.push({
          onCurve: (flag & ON_CURVE) > 0
        });

        if ( flag & REPEAT ) {
          var repeatCount = file.getUint8();
          assert(repeatCount > 0);
          i += repeatCount;
          while( repeatCount-- ) {
            flags.push(flag);
            points.push({
              onCurve: (flag & ON_CURVE) > 0
            });
          }
        }
      }

      function readCoords(name, byteFlag, deltaFlag, min, max) {
        var value = 0;

        for( var i = 0; i < numPoints; i++ ) {
          var flag = flags[i];
          if ( flag & byteFlag ) {
            if ( flag & deltaFlag ) {
              value += file.getUint8();
            } else {
              value -= file.getUint8();
            }
          } else if ( ~flag & deltaFlag ) {
            value += file.getInt16();
          } else {
            // value is unchanged.
          }

          points[i][name] = value;
        }
      }

      readCoords("x", X_IS_BYTE, X_DELTA, glyph.xMin, glyph.xMax);
      readCoords("y", Y_IS_BYTE, Y_DELTA, glyph.yMin, glyph.yMax);
    },

    readCompoundGlyph: function(file, glyph) {
      var ARG_1_AND_2_ARE_WORDS    = 1,
        ARGS_ARE_XY_VALUES       = 2,
        ROUND_XY_TO_GRID         = 4,
        WE_HAVE_A_SCALE          = 8,
        // RESERVED              = 16
        MORE_COMPONENTS          = 32,
```

```
    WE_HAVE_AN_X_AND_Y_SCALE = 64,
    WE_HAVE_A_TWO_BY_TWO     = 128,
    WE_HAVE_INSTRUCTIONS     = 256,
    USE_MY_METRICS           = 512,
    OVERLAP_COMPONENT        = 1024;

glyph.type = "compound";
glyph.components = [];

var flags = MORE_COMPONENTS;
while( flags & MORE_COMPONENTS ) {
    var arg1, arg2;

    flags = file.getUint16();
    var component = {
        glyphIndex: file.getUint16(),
        matrix: {
            a: 1, b: 0, c: 0, d: 1, e: 0, f: 0
        }
    };

    if ( flags & ARG_1_AND_2_ARE_WORDS ) {
        arg1 = file.getInt16();
        arg2 = file.getInt16();
    } else {
        arg1 = file.getUint8();
        arg2 = file.getUint8();
    }

    if ( flags & ARGS_ARE_XY_VALUES ) {
        component.matrix.e = arg1;
        component.matrix.f = arg2;
    } else {
        component.destPointIndex = arg1;
        component.srcPointIndex = arg2;
    }

    if ( flags & WE_HAVE_A_SCALE ) {
        component.matrix.a = file.get2Dot14();
        component.matrix.d = component.matrix.a;
    } else if ( flags & WE_HAVE_AN_X_AND_Y_SCALE ) {
        component.matrix.a = file.get2Dot14();
        component.matrix.d = file.get2Dot14();
    } else if ( flags & WE_HAVE_A_TWO_BY_TWO ) {
        component.matrix.a = file.get2Dot14();
        component.matrix.b = file.get2Dot14();
        component.matrix.c = file.get2Dot14();
        component.matrix.d = file.get2Dot14();
    }
```

```
            glyph.components.push(component);
        }

        if ( flags & WE_HAVE_INSTRUCTIONS ) {
            file.seek(file.getUint16() + file.tell());
        }
    },

    drawGlyph: function(index, ctx) {

        var glyph = this.readGlyph(index);

        if ( glyph === null || glyph.type !== "simple" ) {
            return false;
        }

        var p = 0,
            c = 0,
            first = 1;

        while (p < glyph.points.length) {
            var point = glyph.points[p];
            if ( first === 1 ) {
                ctx.moveTo(point.x, point.y);
                first = 0;
            } else {
                ctx.lineTo(point.x, point.y);
            }

            if ( p === glyph.contourEnds[c] ) {
                c += 1;
                first = 1;
            }

            p += 1;
        }

        return true;
    }
};


function ShowTtfFile(arrayBuffer)
{
    var font = new TrueTypeFont(arrayBuffer);

    var width = font.xMax - font.xMin;
    var height = font.yMax - font.yMin;
    var scale = 64 / font.unitsPerEm;
```

```
    var container = document.getElementById("font-container");

    while(container.firstChild) {
        container.removeChild(container.firstChild);
    }

    for( var i = 0; i < font.length; i++ ) {
        var canvas = document.createElement("canvas");
        canvas.style.border = "1px solid gray";
        canvas.width = width * scale;
        canvas.height = height * scale;
        var ctx = canvas.getContext("2d");
        ctx.fillStyle = "#FFFFFF";
        ctx.fillRect(0,0,200,200);
        ctx.scale(scale, -scale);
        ctx.translate(-font.xMin, -font.yMin - height);
        ctx.fillStyle = "#000000";
        ctx.beginPath();
        if (font.drawGlyph(i, ctx)) {
            ctx.fill();
            container.appendChild(canvas);
            if (i > 3 && i < 94)
            {
                var name = (i.toString()).concat(".png");
                canvas.toBlob(function(blob){
                  saveAs(blob, name);
                 });
            }
        }

    }

}

var dropTarget = document.getElementById("dropTarget");
dropTarget.ondragover = function(e) {
    e.preventDefault();
};
dropTarget.ondrop = function(e) {
    e.preventDefault();

    if (!e.dataTransfer || !e.dataTransfer.files) {
        alert("Your browser didn't include any files in the drop event");
        return;
    }

    var reader = new FileReader();
    reader.readAsArrayBuffer(e.dataTransfer.files[0]);
    reader.onload = function(e) {
        ShowTtfFile(reader.result);
```

```
    };

};
</script>
</body>
</html>
```

seperate.py

```
#!/usr/bin/python
import os
path = '/Users/Alisha/Downloads/'
output = '/Users/Alisha/Desktop/Project/Train/'

for x in range(93,3,-1):
        filepath = output + str(x) + "/";
        os.system("mkdir "+ filepath);
        for i in os.listdir(path):
                if os.path.isfile(os.path.join(path,i)) and str(x) in i:
                        os.system("mv " + path + "'" + i + "'"+" " + filepath);
```

main.java

```
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileFilter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.TermCriteria;
import org.opencv.ml.Ml;
import org.opencv.ml.SVM;

import processing.core.PConstants;
import processing.core.PImage;

public class main {
```

```
	public static void main(String[] args) {
		/******************** Training **********************************/
		BufferedImage[][] Train = new BufferedImage[52][4];
		int numclusters = 40;
		int numofFeatures = 32;
		File folder = new File("/Users/Alisha/Desktop/Project/Train/");
		File[] listOfFiles = folder.listFiles();
		int[] howmanyperchar = new int[208];

		FileFilter fileFilter = new FileFilter() {
			public boolean accept(File file) {
				return file.isDirectory();
			}
		};

		listOfFiles = folder.listFiles(fileFilter);

		int filenum = 0;
		int timagenum = 0;
		loadOpenCVLibrary();
		Mat filename = new Mat(208, 1, CvType.CV_32S);
		ArrayList<Feature> all = new ArrayList<Feature>();
		int count = -1; // the number of features per char
		for (File file : listOfFiles) {
			File read = new File(file.toString());
			File[] images = read.listFiles();
			int imagenum = 0;
			for (File image : images) {
				filename.put(timagenum, 0, Integer.parseInt(image.toString().split("/")[6]));
				// Get each file into buffered image
				try {
					Train[filenum][imagenum] = ImageIO.read(image);
				} catch (IOException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
				BlackWhite.invertImage(Train[filenum][imagenum]);
				PImage now = getAsImage(Train[filenum][imagenum]);
				ArrayList<Feature> featuremade = Sifter.runSift(now,
Integer.toString(imagenum));
				if (featuremade != null) {
					all.addAll(featuremade);
				}
				imagenum++;
				howmanyperchar[timagenum] = featuremade.size();
				timagenum++;
			}
			filenum++;
		}
```

```
        Mat descriptors = new Mat(all.size(), all.get(0).descriptor.length, CvType.CV_32F);
        int featurenum = 0;
        for (Feature f : all) {
                for (int i = 0; i < f.descriptor.length; i++) {
                        descriptors.put(featurenum, i, f.descriptor[i]);
                }
                featurenum++;
        }
        Mat labels = new Mat();
        TermCriteria criteria1 = new TermCriteria(TermCriteria.COUNT, 100, 1);
        Mat centers = new Mat();
        Core.kmeans(descriptors, numclusters, labels, criteria1, 1,
Core.KMEANS_PP_CENTERS, centers);
        System.out.println(centers.size());
        // System.out.println(labels.size());
        // System.out.println(all.size());
        List<Mat> clustermade = Cluster.showClusters(descriptors, labels, centers);

        // for each char send in a feature of char x 50 array
        Mat trainchar = new Mat(208, numclusters, CvType.CV_32F);
        int index = 0; // where in labels array
        for (int i = 0; i < 208; i++) {
                for (int j = 0; j < howmanyperchar[i]; j++) {
                        if (labels.get(0, index) == null) {
                                continue;
                        }
                        double[] current = trainchar.get(i, (int) labels.get(0, index)[0]);
                        trainchar.put(i, (int) labels.get(0, index)[0], current[0]++);
                        index++;
                }
        }
        // Normalize matrix
        for (int i = 0; i < trainchar.rows(); i++) {
                count = 0;
                for (int j = 0; j < trainchar.cols(); j++) {
                        count += trainchar.get(i, j)[0];
                }
                if (count != 0) {
                        for (int j = 0; j < trainchar.cols(); j++) {
                                double prev = trainchar.get(i, j)[0] / count;
                                trainchar.put(i, j, prev);
                        }
                }
        }

        SVM classifier = SVM.create();
        TermCriteria criteria = new TermCriteria(TermCriteria.EPS + TermCriteria.MAX_ITER,
100, 0.1);
        classifier.setKernel(SVM.LINEAR);
```

```
		classifier.setType(SVM.C_SVC);
		classifier.setGamma(0.5);
		classifier.setNu(0.5);
		classifier.setC(1);
		classifier.setTermCriteria(criteria);
		// data is N x 64 trained data Mat , labels is N x 1 label Mat with
		// integer values;
		classifier.train(trainchar, Ml.ROW_SAMPLE, filename);

		/*
		 * // Writing the files
		 *
		 * File outputfile = new
		 * File("/Users/Alisha/Desktop/Project/Train/trial.png"); try {
		 * ImageIO.write(Train[0][0], "png", outputfile); } catch (IOException
		 * e) { // TODO Auto-generated catch block e.printStackTrace(); }
		 */

		/********************** Testing *********************************/
		// Working with an image
		BufferedImage img = null;
		try {
			img = ImageIO.read(new File("/Users/Alisha/Desktop/quote.jpg"));
		} catch (

		IOException e) {
		}
		// Convert to black and white
		BlackWhite.invertImage(img);
		// Cut image into the the individual letters //[ytop, ybottom, xleft,
		// xright]
		int[][] letters = BreakImage.breakimage_init(img);
		BufferedImage[] indivletters = new BufferedImage[letters.length]; // contains

					// the

					// images

		int iterator = 0;
		for (int i = 0; i < letters.length; i++) {
			int height = letters[i][1] - letters[i][0];
			int width = letters[i][3] - letters[i][2];
			int x = letters[i][2];
			int y = letters[i][0];
			if (y + height == img.getHeight() || x + width == img.getWidth())
				continue;
			Rectangle rect = new Rectangle(letters[i][2] - 1, letters[i][0] - 1, letters[i][3] -
letters[i][2] + 3,
					letters[i][1] - letters[i][0] + 3);
			indivletters[iterator] = cropImage(img, rect);
```

```
                iterator++;
        }
        int j = 0;
        for (BufferedImage im : indivletters) {
                File outputfile = new File("/Users/Alisha/Desktop/Project/Trial/trial" +
Integer.toString(j) + ".png");
                j++;
                try {
                        ImageIO.write(im, "png", outputfile);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        for (BufferedImage im : indivletters) {
                PImage testim = getAsImage(im);
                ArrayList<Feature> testfeature = Sifter.runSift(testim, 5, .5f, 2, 8, 16, 64, "test");
                // System.out.println(testfeature.size());
                File outputfile = new File("/Users/Alisha/Desktop/Project/trial.png");
                try {
                        ImageIO.write(im, "png", outputfile);
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                // Creating the descriptors array
                Mat tdescriptors;
                if (testfeature.size() == 0) {
                        tdescriptors = new Mat(1, numofFeatures, CvType.CV_32F);
                } else {
                        tdescriptors = new Mat(testfeature.size(),
testfeature.get(0).descriptor.length, CvType.CV_32F);
                        int tfeaturenum = 0;
                        for (Feature f : testfeature) {
                                for (int i = 0; i < f.descriptor.length; i++) {
                                        tdescriptors.put(tfeaturenum, i, f.descriptor[i]);
                                }
                                tfeaturenum++;
                        }
                }

                Mat finaltest = new Mat(1, numclusters, CvType.CV_32F);
                for (int i = 0; i < tdescriptors.rows(); i++) {
                        int classifycenter = Cluster.findclosest(tdescriptors.row(i), centers);
                        finaltest.put(0, classifycenter, finaltest.get(0, classifycenter)[0] + 1);
                }
                float anwser = classifier.predict(finaltest);
                translate((int) anwser);
```

```
        }
    }

    public static PImage getAsImage(BufferedImage now) {
        try {
            BufferedImage bimg = now;
            PImage img = new PImage(bimg.getWidth(), bimg.getHeight(),
PConstants.ARGB);
            bimg.getRGB(0, 0, img.width, img.height, img.pixels, 0, img.width);
            img.updatePixels();
            return img;
        } catch (Exception e) {
            System.err.println("Can't create image from buffer");
            e.printStackTrace();
        }
        return null;
    }

    private static void loadOpenCVLibrary() {
        // all opencv libs must be copied to OpenCV_lib in the project workspace
        File folder = new File("/Users/Alisha/Desktop/opencv-3.1.0/lib");
        File[] listOfFiles = folder.listFiles();

        for (int i = 0; i < listOfFiles.length; i++) {
            if (listOfFiles[i].isFile() && listOfFiles[i].getName().endsWith(".dylib")) {
                File lib = new File("/Users/Alisha/Desktop/opencv-3.1.0/lib/" +
listOfFiles[i].getName());
                System.load(lib.getAbsoluteFile().toString());
            }
        }
    }

    private static BufferedImage cropImage(BufferedImage src, Rectangle rect) {
        BufferedImage dest = src.getSubimage(rect.x, rect.y, rect.width, rect.height);
        return dest;
    }

    private static void translate(int i) {
        switch (i) {
        case 4:
            System.out.println("!");
            break;
        case 5:
            System.out.println("");
            break;
        case 6:
            System.out.println("#");
            break;
        case 7:
```

```
            System.out.println("$");
            break;
case 8:
            System.out.println("%");
            break;
case 9:
            System.out.println("&");
            break;
case 10:
            System.out.println("\'");
            break;
case 11:
            System.out.println("(");
            break;
case 12:
            System.out.println(")");
            break;
case 13:
            System.out.println("*");
            break;
case 14:
            System.out.println("+");
            break;
case 15:
            System.out.println(",");
            break;
case 16:
            System.out.println("-");
            break;
case 17:
            System.out.println(".");
            break;
case 18:
            System.out.println("/");
            break;
case 19:
            System.out.println("0");
            break;
case 20:
            System.out.println("1");
            break;
case 21:
            System.out.println("2");
            break;
case 22:
            System.out.println("3");
            break;
case 23:
            System.out.println("4");
            break;
```

```
case 24:
        System.out.println("5");
        break;
case 25:
        System.out.println("6");
        break;
case 26:
        System.out.println("7");
        break;
case 27:
        System.out.println("8");
        break;
case 28:
        System.out.println("9");
        break;
case 29:
        System.out.println(":");
        break;
case 30:
        System.out.println(";");
        break;
case 31:
        System.out.println("<");
        break;
case 32:
        System.out.println("=");
        break;
case 33:
        System.out.println(">");
        break;
case 34:
        System.out.println("?");
        break;
case 35:
        System.out.println("@");
        break;
case 36:
        System.out.println("A");
        break;
case 37:
        System.out.println("B");
        break;
case 38:
        System.out.println("C");
        break;
case 39:
        System.out.println("D");
        break;
case 40:
        System.out.println("E");
```

```
            break;
case 41:
            System.out.println("F");
            break;
case 42:
            System.out.println("G");
            break;
case 43:
            System.out.println("H");
            break;
case 44:
            System.out.println("I");
            break;
case 45:
            System.out.println("J");
            break;
case 46:
            System.out.println("k");
            break;
case 47:
            System.out.println("L");
            break;
case 48:
            System.out.println("M");
            break;
case 49:
            System.out.println("N");
            break;
case 50:
            System.out.println("O");
            break;
case 51:
            System.out.println("P");
            break;
case 52:
            System.out.println("Q");
            break;
case 53:
            System.out.println("R");
            break;
case 54:
            System.out.println("S");
            break;
case 55:
            System.out.println("T");
            break;
case 56:
            System.out.println("U");
            break;
case 57:
```

```
            System.out.println("V");
            break;
case 58:
            System.out.println("W");
            break;
case 59:
            System.out.println("X");
            break;
case 60:
            System.out.println("Y");
            break;
case 61:
            System.out.println("Z");
            break;
case 62:
            System.out.println("[");
            break;
case 63:
            System.out.println("\\");
            break;
case 64:
            System.out.println("]");
            break;
case 65:
            System.out.println("^");
            break;
case 66:
            System.out.println("_");
            break;
case 67:
            System.out.println("\'");
            break;
case 68:
            System.out.println("a");
            break;
case 69:
            System.out.println("b");
            break;
case 70:
            System.out.println("c");
            break;
case 71:
            System.out.println("d");
            break;
case 72:
            System.out.println("e");
            break;
case 73:
            System.out.println("f");
            break;
```

```
case 74:
        System.out.println("g");
        break;
case 75:
        System.out.println("h");
        break;
case 76:
        System.out.println("i");
        break;
case 77:
        System.out.println("j");
        break;
case 78:
        System.out.println("k");
        break;
case 79:
        System.out.println("l");
        break;
case 80:
        System.out.println("m");
        break;
case 81:
        System.out.println("n");
        break;
case 82:
        System.out.println("o");
        break;
case 83:
        System.out.println("p");
        break;
case 84:
        System.out.println("q");
        break;
case 85:
        System.out.println("r");
        break;
case 86:
        System.out.println("s");
        break;
case 87:
        System.out.println("t");
        break;
case 88:
        System.out.println("u");
        break;
case 89:
        System.out.println("v");
        break;
case 90:
        System.out.println("w");
```

```
                break;
        case 91:
                System.out.println("x");
                break;
        case 92:
                System.out.println("y");
                break;
        case 93:
                System.out.println("z");
                break;
        default:
                System.out.println("not found "+ Integer.toString(i));
        }

    }
}
```

BlackWhite.java

```
import java.awt.*;
import java.awt.image.BufferedImage;

class BlackWhite {

    public static void invertImage(BufferedImage inputFile) {

        int MONO_THRESHOLD = 510; // Trial and error
        for (int x = 0; x < inputFile.getWidth(); x++) {
                for (int y = 0; y < inputFile.getHeight(); y++) {
                        int rgba = inputFile.getRGB(x, y);
                        Color col = new Color(rgba, true);
                        if (col.getRed() + col.getGreen() + col.getBlue() < MONO_THRESHOLD)
                                col = new Color(0,0,0);
                        else
                                col = new Color(255, 255, 255);
                        inputFile.setRGB(x, y, col.getRGB());
                }
        }
    }
        }
```

Sifter.java

```
/**
 * Sifter.java
 *
 * Part of the library for the class 10-615 Art That Learns taught in
 * Carnegie Mellon University on Spring 2009 semester
 *
 */


// The Java SIFT library
import mpi.cbg.fly.*;
// Processing library
import processing.core.*;

// Java Libraries
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Vector;

/**
 * Class that handles the SIFT wrappers. SIFT is explained in:
 * <a href="http://en.wikipedia.org/wiki/Scale-invariant_feature_transform">Scale Invariant
Feature Transform</a>
 *
 * Java SIFT code is taken from <a href="http://fly.mpi-cbg.de/~saalfeld/javasift.html>Stephan
Saalfeld</a>'s
 * page and stripped down to be callable from other Java programs without needing ImageJ.
 *
 *
 *
 * @author Barkin Aygun
 *
 *
 * @see <a
href="http://dev.processing.org/reference/core/javadoc/processing/core/PApplet.html">
 * processing.core.PApplet</a>
 *
 * @usage Library
 */
public class Sifter {
```

```
// steps
private static int steps = 10;
// initial sigma
private static float initial_sigma = 1.6f;
// feature descriptor size
private static int fdsize = 2;
// feature descriptor orientation bins
private static int fdbins = 8;
// size restrictions for scale octaves, use octaves < max_size and > min_size only
private static int min_size = 16;
private static int max_size = 1024;

/**
 * For using PImages directly from Processing for SIFT
 * @param image          PImage to be converted
 * @return FloatArray2D
 */
private static FloatArray2D ConvertImage(PImage image) {
   // Set image to grayscale and load pixels for reading
   int count = 0;
   FloatArray2D result;
   PImage bwimage = image;
   bwimage.filter(PImage.GRAY);
   bwimage.loadPixels();

   result = new FloatArray2D(bwimage.width, bwimage.height);
   for (int x = 0; x < bwimage.height; x++) {
            for (int y = 0; y < bwimage.width; y++) {
                     result.data[count] = bwimage.pixels[count++] / 255.0f;
            }
   }

   return result;
}

/**
 * Runs sift using default parameters on a given PImage
 * @param image PImage to run Sift on
 * @return Vector<Feature> Set of features found on the image
 *
 * @see Feature
 */
public static ArrayList<Feature> runSift(PImage image, String imageId) {
   return runSift(image, steps, initial_sigma, fdsize, fdbins, min_size, max_size, imageId);
}

/**
 * Runs sift on a given PImage with given parameters
 *
 * @param image PImage to run Sift on
```

```
         * @param steps Number of steps
         * @param initial_sigma Initial Blur Factor
         * @param fdsize Feature Descriptor Size
         * @param fdbins Number of Feature Descriptor Bins
         * @param min_size Minimum size for a feature
         * @param max_size Maximum size for a feature
         * @return Vector of Features
         *
         * @see Feature
         */
        public static ArrayList<Feature> runSift(PImage image, int steps, float initial_sigma,
                                                 int fdsize, int fdbins, int min_size, int
max_size, String imageId) {
            Vector<mpi.cbg.fly.Feature> fs;
            FloatArray2DSIFT sift = new FloatArray2DSIFT( fdsize, fdbins );
            FloatArray2D fa = ConvertImage(image);
            Filter.enhance(fa, 1.0f);

            fa = Filter.computeGaussianFastMirror(fa, (float )Math.sqrt(initial_sigma * initial_sigma -
0.25) );

            sift.init(fa, steps, initial_sigma, min_size, max_size);
            fs = sift.run(max_size);

            ArrayList<Feature> fs1 = new ArrayList<Feature>(fs.size());
            for (mpi.cbg.fly.Feature f : fs) {
                    Feature tempf = new Feature(f.scale, f.orientation, f.location, f.descriptor);
                    tempf.imageId = imageId;
                    fs1.add(tempf);
            }

            return fs1;
        }

        public static int[] matchFeatures(ArrayList<Feature> set_one, ArrayList<Feature> set_two,
float threshold) {
            int[] matches = new int[set_one.size()];
            int i = 0;
            for (Feature f : set_one) {
                    matches[i++] = closestMatch(f, set_two, threshold);
            }
            return matches;
        }

        public static int closestMatch(Feature f1, ArrayList<Feature> possible_matches, float
threshold) {
            float maxMatch = 0;
            float secondMax = 0;
            float desDist = 0;
            int maxIndex = 0;
```

```
        int curIndex = 0;
        for (Feature f : possible_matches) {
                desDist = f1.descriptorDistance(f);
                //if (desDist < threshold) {curIndex++; continue;};
                if (desDist > maxMatch) {
                        secondMax = maxMatch;
                        maxIndex = curIndex;
                        maxMatch = desDist;
                }
                curIndex++;
        }
        System.out.print(maxMatch + " against " + secondMax);
        if (maxMatch > 0.8f && maxMatch * 0.8f < secondMax) {
                System.out.println(" succeeds");
                return maxIndex;
        }
        else {
                System.out.println(" fails");
                return -1;
        }
    }

    public static void saveFeatures(ArrayList<Feature> fs, String filename) {
        FileWriter fw;
        try {
                fw = new FileWriter(filename);
                for (Feature f : fs) {
                        fw.write(f.toString());
                }
                fw.close();
        } catch (IOException e) {
                System.out.println("File can not be opened for write: " + filename);
        }
    }

    public static ArrayList<Feature> loadFeatures(String filename) {
        BufferedReader in;
        String line;
        ArrayList<Feature> fs = new ArrayList<Feature>();
        try {
                in = new BufferedReader(new FileReader(filename));
                while ((line = in.readLine()) != null) {
                        fs.add(new Feature(line));
                }
        } catch (IOException e) {
                System.out.println("File can not be opened for read: " + filename);
        }
        return fs;
    }
}
```

Feature.java

```
import java.io.Serializable;
import java.util.StringTokenizer;

/**
 * SIFT feature container
 */
public class Feature implements Comparable< Feature >, Serializable
{
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	public float scale;
	public float orientation;
	public float[] location;
	public float[] descriptor;
	public String imageId;
	public String matchId;
	public boolean matched;

	/** Dummy constructor for Serialization to work properly. */
	public Feature() {}

	public Feature( float s, float o, float[] l, float[] d )
	{
	   scale = s;
	   orientation = o;
	   location = l;
	   descriptor = d;
	   matched = false;
	}

	public Feature( String encodedFeature) {
	   StringTokenizer st = new StringTokenizer(encodedFeature, ",");
	   String nt;
	   int dcount = 0;

	   nt = st.nextToken();
	   imageId = nt;

	   nt = st.nextToken();
	   scale = Float.parseFloat(nt);

	   nt = st.nextToken();
	   orientation = Float.parseFloat(nt);
```

```
            location = new float[2];
            location[0] = Float.parseFloat(st.nextToken());
            location[1] = Float.parseFloat(st.nextToken());

            descriptor = new float[st.countTokens()];
            while(st.hasMoreTokens()) {
                    descriptor[dcount++] = Float.parseFloat(st.nextToken());
            }
            matched = false;
        }

        /**
         * comparator for making Features sortable
         * please note, that the comparator returns -1 for
         * this.scale &gt; o.scale, to sort the features in a descending order
         */
        public int compareTo( Feature f )
        {
            return scale < f.scale ? 1 : scale == f.scale ? 0 : -1;
        }

        public float descriptorDistance( Feature f )
        {
            float d = 0;
            for ( int i = 0; i < descriptor.length; ++i )
            {
                    float a = descriptor[ i ] - f.descriptor[ i ];
                    d += a * a;
            }
            return ( float )Math.sqrt( d );
        }

        public String toString() {
            StringBuilder sb = new StringBuilder();
            sb.append(this.imageId + ",");
            sb.append(this.scale + ",");
            sb.append(this.orientation + ",");
            sb.append(this.location[0] + "," + this.location[1] + ",");
            for (float d : this.descriptor) {
                    sb.append(d + ",");
            }
            sb.append("\n");
            return sb.toString();
        }
}
```

Cluster.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.TermCriteria;

import java.lang.Double;


public class Cluster {

	public static List<Mat> showClusters (Mat cutout, Mat labels, Mat centers) {
	  centers.convertTo(centers, CvType.CV_8UC1, 255.0);
	  //System.out.println(centers.size());
	  //centers.reshape(3);

	  List<Mat> clusters = new ArrayList<Mat>();
	  for(int i = 0; i < centers.rows(); i++) {
		clusters.add(Mat.zeros(cutout.size(), cutout.type()));
	  }

	  Map<Integer, Integer> counts = new HashMap<Integer, Integer>();
	  for(int i = 0; i < centers.rows(); i++) counts.put(i, 0);

	  int rows = 0;
	  for(int y = 0; y < cutout.rows(); y++) {
		for(int x = 0; x < cutout.cols(); x++) {
			if (labels.get(rows,0) == null)
			{
				//System.out.println("skipped");
				continue;
			}
			//System.out.println("got center");
			int label = (int)labels.get(rows, 0)[0];
			int r = (int)centers.get(label, 2)[0];
			int g = (int)centers.get(label, 1)[0];
			int b = (int)centers.get(label, 0)[0];
			counts.put(label, counts.get(label) + 1);
			clusters.get(label).put(y, x, b, g, r);
			rows++;
		}
	  }
	  //System.out.println(counts);
```

```
        return clusters;
    }

    public static int findclosest(Mat row, Mat centers)
    {
        double ldistance = Double.POSITIVE_INFINITY;
        int centernum = 0;
        for (int i = 0; i < centers.rows(); i++)
        {
                int distance = 0;
                for (int j = 0; j < centers.cols(); j++)
                {
                        distance += Math.pow(centers.get(i, j)[0] - row.get(0, j)[0], 2);
                }
                Math.sqrt(distance);
                if (distance < ldistance)
                {
                        centernum = i;
                        ldistance = distance;
                }
        }
        return centernum;

    }
}
```

BreakImage.java

```
import java.awt.image.BufferedImage;
import java.util.ArrayList;

public class BreakImage {
    static BufferedImage Input; // image being examined
    static int[][] rowArray;
    static int[][] letterArray;
    static int lines; // number of lines of text
    static int letters; //number of letters
    static int tletters; //total letters count

    // Main function called to break image
    public static int[][] breakimage_init(BufferedImage input) {
        Input = input;
        rowBreak();
        columnBreak();
        return letterArray;
```

```
}

// Break image into rows first
private static void rowBreak() {
   lines = 0;
   int[] sumr = new int[Input.getHeight()];
   // get number of lines by finding where positive value begins
   for (int y = 0; y < Input.getHeight(); y++) {
            for (int x = 0; x < Input.getWidth(); x++) {
                     if (Input.getRGB(x, y) != -1)
                              sumr[y]++;
            }
            if (sumr[y] != 0 && (y == 0 || sumr[y - 1] == 0))
                     lines++;
   }

   rowArray = new int[lines][2];
   int iterator = 0;

   // get exact location of the lines
   for (int y = 0; y < Input.getHeight(); y++) {
            if (sumr[y] != 0 && (y == 0 || sumr[y - 1] == 0))
                     rowArray[iterator][0] = y;
            else if ((y + 1 == Input.getHeight() || sumr[y + 1] == 0) && sumr[y] != 0) {
                     rowArray[iterator][1] = y;
                     iterator++;
            }
   }

   // Output system
   /*System.out.println(lines);
   for (iterator = 0; iterator < lines; iterator++) {
            System.out.print(rowArray[iterator][0]);
            System.out.print(" , ");
            System.out.println(rowArray[iterator][1]);
   }*/
}

private static void columnBreak() {
   tletters = 0;
   for (int i = 0; i < rowArray.length; i++) {
            int[] sumc = new int[Input.getWidth()];
            letters = 0;
            for (int x = 0; x < Input.getWidth(); x++) {
                     for (int y = rowArray[i][0]; y <= rowArray[i][1]; y++) {
                              if (Input.getRGB(x, y) != -1)
                                       sumc[x]++;
                     }
                     if (sumc[x] != 0 && (x == 0 || sumc[x - 1] == 0))
                              letters++;
```

```
                }

                // puts total count in tletters
                tletters += letters;
        }
        System.out.println(tletters);
        letterArray = new int[tletters][4];//[ytop, ybottom, xleft, xright]
        int iterator = 0;

        // get each letter coordinates
        for (int i = 0; i < rowArray.length; i++) {
                int[] sumc = new int[Input.getWidth()];
                for (int x = 0; x < Input.getWidth(); x++) {
                        for (int y = rowArray[i][0]; y <= rowArray[i][1]; y++) {
                                if (Input.getRGB(x, y) != -1)
                                        sumc[x]++;
                        }

                        if (sumc[x] != 0 && (x == 0 || sumc[x - 1] == 0))
                        {
                                letterArray[iterator][0] = rowArray[i][0];
                                letterArray[iterator][1] = rowArray[i][1];
                                letterArray[iterator][2] = x;
                        }
                        else if (x == 0)
                                continue;
                        else if (sumc[x] == 0 && sumc[x - 1] != 0)
                        {
                                letterArray[iterator][3] = x-1;
                                iterator++;
                        }
                }
        }

        /*for (iterator = 0; iterator < tletters; iterator++) {
                System.out.print(letterArray[iterator][0]);
                System.out.print(" , ");
                System.out.print(letterArray[iterator][1]);
                System.out.print(" , ");
                System.out.print(letterArray[iterator][2]);
                System.out.print(" , ");
                System.out.println(letterArray[iterator][3]);
        }*/

    }

}
```

REFERENCES

[1] S. Hanov, "Let's read a Truetype font file from scratch," Steve Hanov, 20 April 2015. [Online]. Available: http://stevehanov.ca/blog/index.php?id=143. [Accessed 20 March 2016].

[2] CMU, "Sifter.java," preplab.googlecode.com, 01 01 2009. [Online]. Available: http://preplab.googlecode.com/svn/trunk/src/imagelib/Sifter.java. [Accessed 02 04 2016].

[3] G. Bradski, "opencv_library," Dr. Dobb's Journal of Software Tools, n/a, 2008.

[4] J. Austen, "Pride and Prejudice," Modern Library, New York, 1995.

[5] A. Vedaldi and A. Zisserman, "VGG Convolutional Neural Networks Practical," Oxford AIMS CDT, n/a, 2014.

[6] S. VonLooy, J. Criss and W. Joe, "2013 Annual Disability Status Report," DisabilityStatistics.org, Ithaca, 2014.

[7] X.R. Chen and A.L. Yuille, "Detecting and Reading Text in Natural Scenes," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR'04),* pp. 366-373, 2004.

[8] B. C. Mills and L. J. Weldon, "Reading text from computer screens," *ACM Computing Surveys (CSUR),* vol. 19, no. 4, pp. 329-357, 1987.

Academic Vita of Alisha Rege

Alisha.rege@gmail.com

Education
Major(s) and Minor(s): Computer Engineering
Honors: Computer Science

Thesis Title: From Image to Words: Computer Vision and Machine Learning
Thesis Supervisor: Dr. Robert Collins

Work Experience

June 2015 – August 2015
Undergraduate Researcher
- Worked on Machine Learning integrated with Audio Video Processing
- Combined Audio and Visual data to detect talking heads in the REPERE corpus data (French Dataset)
- Analyzed 60 hours of live data to create a program that could analyze mouths on rotated and upright heads

MIT Lincoln Laboratory

Jan 2014 – Jan 2015
Undergraduate Researcher
- Researched Oculus Rift to develop a program that allows users to see multiple data points at once and manipulate them in a 3 dimensional atmosphere (with virtual glasses) – the CAVE technology

The Advanced Research Laboratory

May 2014 – Aug 2014
Software Development and Testing Intern
- Developed a statistical model/machine learned model to find anomalies in journaled emails
- Created an E2E notification system that saved the company from law suits and protected the information of the customers.
- Became proficient in using Scope to query structured streams and manipulate data.

Microsoft

June 2012- June 2013
Self-made App Designer – Speech2Sign

- Self-taught Objective C and American Sign Language
- Designed app that uses Siri to help hearing impaired people and students
- Won many awards: 1st place in Intel Regional completion and 2nd place in States competition

Professional Memberships: Eta Kappa Nu, Tau Beta Pi

Community Service Involvement: Tutoring 2 hours every week for 2 years

International Education (including service-learning abroad):

Language Proficiency:

- English
- Marathi
- Hindi
- French