**CODE:**

**OCREngine.java**

import java.awt.Color;

import java.awt.Container;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.BoxLayout;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import net.sourceforge.tess4j.Tesseract;

public class OCREngine extends JFrame

{

private static final long serialVersionUID = 1L;

JFrame frame=null;

JLabel extractLabel=null,spellCheckedLabel=null,imageLabel=null,picLabel=null,

picLabel1=null;

JTextField extractedWord=null, spellCheckedText=null,imagePath=null;

JButton spellCheck=null,display=null,extractText=null;

BufferedImage image=null;

public static JFrame mainframe;

public void printPixelARGB(int pixel)

{

int alpha = (pixel >> 24) & 0xff;

int red = (pixel >> 16) & 0xff;

int green = (pixel >> 8) & 0xff;

int blue = (pixel) & 0xff;

System.out.println("argb: " + alpha + ", " + red + ", " + green + ", " + blue);

}

public OCREngine()

{

final Container container=getContentPane();

BoxLayout layout=new BoxLayout(container,BoxLayout.Y\_AXIS);

container.setLayout(layout);

JPanel PimagePath;

final JPanel Pimage;

final JPanel Pextract;

final JPanel Pspell;

PimagePath=new JPanel();

Pimage=new JPanel();

Pextract=new JPanel();

Pspell=new JPanel();

imageLabel=new JLabel("Enter the path of the image file");

imagePath=new JTextField(40);

display=new JButton("Display image");

picLabel=new JLabel();

extractText=new JButton("Extract Text");

extractLabel=new JLabel("Extracted word");

extractedWord=new JTextField(20);

extractedWord.setEditable(true);

spellCheck=new JButton("Check Spelling");

spellCheckedLabel=new JLabel("Correct Spelling");

spellCheckedText=new JTextField(20);

spellCheckedText.setEditable(false);

PimagePath.add(imageLabel);

PimagePath.add(imagePath);

PimagePath.add(display);

Pextract.add(extractLabel);

Pextract.add(extractedWord);

Pextract.add(spellCheck);

Pspell.add(spellCheckedLabel);

Pspell.add(spellCheckedText);

container.add(PimagePath);

display.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent ae)

{

try

{

String str=null;

//"image" is an object of BufferedImage

if((str=imagePath.getText()).isEmpty())

{

JOptionPane.showMessageDialog(null,"Enter a valid

imagepath","Enter a valid imagepath",JOptionPane.ERROR\_MESSAGE);

}

else

{

image = ImageIO.read(new File(imagePath.getText()));

System.out.println(imagePath.getText());

Pimage.add(picLabel1=new JLabel(new

ImageIcon(image)));

//Using Canny's Edge Detection

//create the detector

CannyEdgeDetector detector = new CannyEdgeDetector();

//adjust its parameters as desired

detector.setLowThreshold(0.5f);

detector.setHighThreshold(1f);

//apply it to an image

detector.setSourceImage(image);

detector.process();

BufferedImage edges = detector.getEdgesImage();

int w=edges.getWidth();

int h=edges.getHeight();

System.out.println("width, height: " + w + ", " + h);

for(int i=0;i<h;i++)

{

for(int j=0;j<w;j++)

{

System.out.println("x,y: " + j + ", " + i);

int pixel = image.getRGB(j, i);

printPixelARGB(pixel);

System.out.println("");

}

}

//adding edge-detected picture and extractText to Piamge

Pimage.add(picLabel=new JLabel(new ImageIcon(edges)));

Pimage.add(extractText);

//adding all panels to container

container.add(Pimage);

container.add(Pextract);

container.add(Pspell);

revalidate();

//mainframe.repaint();

}

}

catch (IOException ex)

{

ex.printStackTrace();

}

}

});

extractText.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent ae)

{

try

{

Tesseract instance = Tesseract.getInstance();

String result = instance.doOCR(image);

extractedWord.setText(result);

}

catch (Exception ex)

{

ex.printStackTrace();

}

}

});

spellCheck.addActionListener(new ActionListener()

{

public void actionPerformed (ActionEvent ae)

{

try

{

String str=null;

if((str=extractedWord.getText()).isEmpty())

{

JOptionPane.showMessageDialog(null,"Invalid word","Invalid word",JOptionPane.ERROR\_MESSAGE);

}

else

{

JazzySpellChecker jazzySpellChecker = new

JazzySpellChecker();

String line =

jazzySpellChecker.getCorrectedLine(extractedWord.getText());

spellCheckedText.setText(line);

BufferedImage newImage=new

BufferedImage(200,70,BufferedImage.TYPE\_INT\_ARGB);

BufferedImage replacedImage = ImageIO.read(new

File("F:/SpellChecker/res/NewImage.jpg"));

Graphics2D g2d=newImage.createGraphics();

g2d.drawImage(replacedImage,0,0,null);

FontMetrics fm=g2d.getFontMetrics();

g2d.setFont(new Font("Calibri",Font.BOLD,50));

g2d.setColor(Color.BLACK);

int x=(newImage.getWidth()/2)-30;

int y=newImage.getHeight()-20;

g2d.drawString(line,x,y);

g2d.dispose();

JPanel Presult=new JPanel();

Presult.add(picLabel=new JLabel(new

ImageIcon(newImage)));

//adding panel to container

container.add(Presult);

revalidate();

}

}

catch (Exception e)

{

e.printStackTrace();

}

}

}

);

}

public static void main(String [] args) throws Exception

{

OCREngine frame=new OCREngine();

frame.setSize(800, 600);

frame.setVisible(true);

frame.addWindowListener( new WindowAdapter()

{

public void windowClosing(WindowEvent we)

{

System.exit(0);

}

});

}

}

**CannyEdgeDetector.java**

import java.awt.image.BufferedImage;

import java.util.Arrays;

public class CannyEdgeDetector {

private final static float GAUSSIAN\_CUT\_OFF = 0.005f;

private final static float MAGNITUDE\_SCALE = 100F;

private final static float MAGNITUDE\_LIMIT = 1000F;

private final static int MAGNITUDE\_MAX = (int) (MAGNITUDE\_SCALE \* MAGNITUDE\_LIMIT);

private int height;

private int width;

private int picsize;

private int[] data;

private int[] magnitude;

private BufferedImage sourceImage;

private BufferedImage edgesImage;

private float gaussianKernelRadius;

private float lowThreshold;

private float highThreshold;

private int gaussianKernelWidth;

private boolean contrastNormalized;

private float[] xConv;

private float[] yConv;

private float[] xGradient;

private float[] yGradient;

public CannyEdgeDetector() {

lowThreshold = 2.5f;

highThreshold = 7.5f;

gaussianKernelRadius = 2f;

gaussianKernelWidth = 16;

contrastNormalized = false;

}

public BufferedImage getSourceImage() {

return sourceImage;

}

public void setSourceImage(BufferedImage image) {

sourceImage = image;

}

public BufferedImage getEdgesImage() {

return edgesImage;

}

public void setEdgesImage(BufferedImage edgesImage) {

this.edgesImage = edgesImage;

}

public float getLowThreshold() {

return lowThreshold;

}

public void setLowThreshold(float threshold) {

if (threshold < 0) throw new IllegalArgumentException();

lowThreshold = threshold;

}

public float getHighThreshold() {

return highThreshold;

}

public void setHighThreshold(float threshold) {

if (threshold < 0) throw new IllegalArgumentException();

highThreshold = threshold;

}

public int getGaussianKernelWidth() {

return gaussianKernelWidth;

}

public void setGaussianKernelWidth(int gaussianKernelWidth) {

if (gaussianKernelWidth < 2) throw new IllegalArgumentException();

this.gaussianKernelWidth = gaussianKernelWidth;

}

public float getGaussianKernelRadius() {

return gaussianKernelRadius;

}

public void setGaussianKernelRadius(float gaussianKernelRadius) {

if (gaussianKernelRadius < 0.1f) throw new IllegalArgumentException();

this.gaussianKernelRadius = gaussianKernelRadius;

}

public boolean isContrastNormalized() {

return contrastNormalized;

}

public void setContrastNormalized(boolean contrastNormalized) {

this.contrastNormalized = contrastNormalized;

}

// methods

public void process() {

width = sourceImage.getWidth();

height = sourceImage.getHeight();

picsize = width \* height;

initArrays();

readLuminance();

if (contrastNormalized) normalizeContrast();

computeGradients(gaussianKernelRadius, gaussianKernelWidth);

int low = Math.round(lowThreshold \* MAGNITUDE\_SCALE);

int high = Math.round( highThreshold \* MAGNITUDE\_SCALE);

performHysteresis(low, high);

thresholdEdges();

writeEdges(data);

}

// private utility methods

private void initArrays() {

if (data == null || picsize != data.length) {

data = new int[picsize];

magnitude = new int[picsize];

xConv = new float[picsize];

yConv = new float[picsize];

xGradient = new float[picsize];

yGradient = new float[picsize];

}

}

private void computeGradients(float kernelRadius, int kernelWidth) {

float kernel[] = new float[kernelWidth];

float diffKernel[] = new float[kernelWidth];

int kwidth;

for (kwidth = 0; kwidth < kernelWidth; kwidth++) {

float g1 = gaussian(kwidth, kernelRadius);

if (g1 <= GAUSSIAN\_CUT\_OFF && kwidth >= 2) break;

float g2 = gaussian(kwidth - 0.5f, kernelRadius);

float g3 = gaussian(kwidth + 0.5f, kernelRadius);

kernel[kwidth] = (g1 + g2 + g3) / 3f / (2f \* (float) Math.PI \* kernelRadius \* kernelRadius);

diffKernel[kwidth] = g3 - g2;

}

int initX = kwidth - 1;

int maxX = width - (kwidth - 1);

int initY = width \* (kwidth - 1);

int maxY = width \* (height - (kwidth - 1));

for (int x = initX; x < maxX; x++) {

for (int y = initY; y < maxY; y += width) {

int index = x + y;

float sumX = data[index] \* kernel[0];

float sumY = sumX;

int xOffset = 1;

int yOffset = width;

for(; xOffset < kwidth ;) {

sumY += kernel[xOffset] \* (data[index - yOffset] + data[index + yOffset]);

sumX += kernel[xOffset] \* (data[index - xOffset] + data[index + xOffset]);

yOffset += width;

xOffset++;

}

yConv[index] = sumY;

xConv[index] = sumX;

}

}

for (int x = initX; x < maxX; x++) {

for (int y = initY; y < maxY; y += width) {

float sum = 0f;

int index = x + y;

for (int i = 1; i < kwidth; i++)

sum += diffKernel[i] \* (yConv[index - i] - yConv[index + i]);

xGradient[index] = sum;

}

}

for (int x = kwidth; x < width - kwidth; x++) {

for (int y = initY; y < maxY; y += width) {

float sum = 0.0f;

int index = x + y;

int yOffset = width;

for (int i = 1; i < kwidth; i++) {

sum += diffKernel[i] \* (xConv[index - yOffset] - xConv[index + yOffset]);

yOffset += width;

}

yGradient[index] = sum;

}

}

initX = kwidth;

maxX = width - kwidth;

initY = width \* kwidth;

maxY = width \* (height - kwidth);

for (int x = initX; x < maxX; x++) {

for (int y = initY; y < maxY; y += width) {

int index = x + y;

int indexN = index - width;

int indexS = index + width;

int indexW = index - 1;

int indexE = index + 1;

int indexNW = indexN - 1;

int indexNE = indexN + 1;

int indexSW = indexS - 1;

int indexSE = indexS + 1;

float xGrad = xGradient[index];

float yGrad = yGradient[index];

float gradMag = hypot(xGrad, yGrad);

//perform non-maximal supression

float nMag = hypot(xGradient[indexN], yGradient[indexN]);

float sMag = hypot(xGradient[indexS], yGradient[indexS]);

float wMag = hypot(xGradient[indexW], yGradient[indexW]);

float eMag = hypot(xGradient[indexE], yGradient[indexE]);

float neMag = hypot(xGradient[indexNE], yGradient[indexNE]);

float seMag = hypot(xGradient[indexSE], yGradient[indexSE]);

float swMag = hypot(xGradient[indexSW], yGradient[indexSW]);

float nwMag = hypot(xGradient[indexNW], yGradient[indexNW]);

float tmp;

if (xGrad \* yGrad <= (float) 0 /\*(1)\*/

? Math.abs(xGrad) >= Math.abs(yGrad) /\*(2)\*/

? (tmp = Math.abs(xGrad \* gradMag)) >= Math.abs(yGrad

\* neMag - (xGrad + yGrad) \* eMag) /\*(3)\*/

&& tmp > Math.abs(yGrad \* swMag - (xGrad +

yGrad) \* wMag) /\*(4)\*/

: (tmp = Math.abs(yGrad \* gradMag)) >= Math.abs(xGrad

\* neMag - (yGrad + xGrad) \* nMag) /\*(3)\*/

&& tmp > Math.abs(xGrad \* swMag - (yGrad +

xGrad) \* sMag) /\*(4)\*/

: Math.abs(xGrad) >= Math.abs(yGrad) /\*(2)\*/

? (tmp = Math.abs(xGrad \* gradMag)) >= Math.abs(yGrad \* seMag + (xGrad - yGrad) \* eMag) /\*(3)\*/

&& tmp > Math.abs(yGrad \* nwMag + (xGrad - yGrad) \* wMag) /\*(4)\*/

: (tmp = Math.abs(yGrad \* gradMag)) >= Math.abs(xGrad \* seMag + (yGrad - xGrad) \* sMag) /\*(3)\*/

&& tmp > Math.abs(xGrad \* nwMag + (yGrad - xGrad) \* nMag) /\*(4)\*/

) {

magnitude[index] = gradMag >= MAGNITUDE\_LIMIT ? MAGNITUDE\_MAX : (int) (MAGNITUDE\_SCALE \* gradMag);

} else {

magnitude[index] = 0;

}

}

}

}

private float hypot(float x, float y) {

return (float) Math.hypot(x, y);

}

private float gaussian(float x, float sigma) {

return (float) Math.exp(-(x \* x) / (2f \* sigma \* sigma));

}

private void performHysteresis(int low, int high) {

Arrays.fill(data, 0);

int offset = 0;

for (int y = 0; y < height; y++) {

for (int x = 0; x < width; x++) {

if (data[offset] == 0 && magnitude[offset] >= high) {

follow(x, y, offset, low);

}

offset++;

}

}

}

private void follow(int x1, int y1, int i1, int threshold) {

int x0 = x1 == 0 ? x1 : x1 - 1;

int x2 = x1 == width - 1 ? x1 : x1 + 1;

int y0 = y1 == 0 ? y1 : y1 - 1;

int y2 = y1 == height -1 ? y1 : y1 + 1;

data[i1] = magnitude[i1];

for (int x = x0; x <= x2; x++) {

for (int y = y0; y <= y2; y++) {

int i2 = x + y \* width;

if ((y != y1 || x != x1)

&& data[i2] == 0

&& magnitude[i2] >= threshold) {

follow(x, y, i2, threshold);

return;

}

}

}

}

private void thresholdEdges() {

for (int i = 0; i < picsize; i++) {

data[i] = data[i] > 0 ? -1 : 0xff000000;

}

}

private int luminance(float r, float g, float b) {

return Math.round(0.299f \* r + 0.587f \* g + 0.114f \* b);

}

private void readLuminance() {

int type = sourceImage.getType();

if (type == BufferedImage.TYPE\_INT\_RGB || type == BufferedImage.TYPE\_INT\_ARGB) {

int[] pixels = (int[]) sourceImage.getData().getDataElements(0, 0, width, height, null);

for (int i = 0; i < picsize; i++) {

int p = pixels[i];

int r = (p & 0xff0000) >> 16;

int g = (p & 0xff00) >> 8;

int b = p & 0xff;

data[i] = luminance(r, g, b);

}

} else if (type == BufferedImage.TYPE\_BYTE\_GRAY) {

byte[] pixels = (byte[]) sourceImage.getData().getDataElements(0, 0, width, height, null);

for (int i = 0; i < picsize; i++) {

data[i] = (pixels[i] & 0xff);

}

} else if (type == BufferedImage.TYPE\_USHORT\_GRAY) {

short[] pixels = (short[]) sourceImage.getData().getDataElements(0, 0, width, height, null);

for (int i = 0; i < picsize; i++) {

data[i] = (pixels[i] & 0xffff) / 256;

}

} else if (type == BufferedImage.TYPE\_3BYTE\_BGR) {

byte[] pixels = (byte[]) sourceImage.getData().getDataElements(0, 0, width, height, null);

int offset = 0;

for (int i = 0; i < picsize; i++) {

int b = pixels[offset++] & 0xff;

int g = pixels[offset++] & 0xff;

int r = pixels[offset++] & 0xff;

data[i] = luminance(r, g, b);

}

} else {

throw new IllegalArgumentException("Unsupported image type: " + type);

}

}

private void normalizeContrast() {

int[] histogram = new int[256];

for (int i = 0; i < data.length; i++) {

histogram[data[i]]++;

}

int[] remap = new int[256];

int sum = 0;

int j = 0;

for (int i = 0; i < histogram.length; i++) {

sum += histogram[i];

int target = sum\*255/picsize;

for (int k = j+1; k <=target; k++) {

remap[k] = i;

}

j = target;

}

for (int i = 0; i < data.length; i++) {

data[i] = remap[data[i]];

}

}

private void writeEdges(int pixels[]) {

if (edgesImage == null) {

edgesImage = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_ARGB);

}

edgesImage.getWritableTile(0, 0).setDataElements(0, 0, width, height, pixels); }}

**JazzySpellChecker.java**

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import com.swabunga.spell.engine.SpellDictionaryHashMap;

import com.swabunga.spell.engine.Word;

import com.swabunga.spell.event.SpellCheckEvent;

import com.swabunga.spell.event.SpellCheckListener;

import com.swabunga.spell.event.SpellChecker;

import com.swabunga.spell.event.StringWordTokenizer;

import com.swabunga.spell.event.TeXWordFinder;

public class JazzySpellChecker implements SpellCheckListener

{

private SpellChecker spellChecker;

private List<String> misspelledWords;

public List<String> getMisspelledWords(String text)

{

StringWordTokenizer texTok = new StringWordTokenizer(text,

new TeXWordFinder());

spellChecker.checkSpelling(texTok);

return misspelledWords;

}

private static SpellDictionaryHashMap dictionaryHashMap;

static

{

File dict = new File("dictionary/dictionary.txt");

try

{

dictionaryHashMap = new SpellDictionaryHashMap(dict);

}

catch (FileNotFoundException e)

{

e.printStackTrace();

}

catch (IOException e)

{

e.printStackTrace();

}

}

private void initialize()

{

spellChecker = new SpellChecker(dictionaryHashMap);

spellChecker.addSpellCheckListener(this);

}

public JazzySpellChecker()

{

misspelledWords = new ArrayList<String>();

initialize();

}

public String getCorrectedLine(String line)

{

List<String> misSpelledWords = getMisspelledWords(line);

for (String misSpelledWord : misSpelledWords){

List<String> suggestions = getSuggestions(misSpelledWord);

if (suggestions.size() == 0)

continue;

String bestSuggestion = suggestions.get(0);

line = line.replace(misSpelledWord, bestSuggestion);

}

return line;

}

public String getCorrectedText(String line)

{

StringBuilder builder = new StringBuilder();

String[] tempWords = line.split(" ");

for (String tempWord : tempWords)

{

if (!spellChecker.isCorrect(tempWord))

{

List<Word> suggestions = spellChecker.getSuggestions(tempWord, 0);

if (suggestions.size() > 0)

{

builder.append(spellChecker.getSuggestions(tempWord, 0).get(0).toString());

}

else

builder.append(tempWord);

}

else

{

builder.append(tempWord);

}

builder.append(" ");

}

return builder.toString().trim();

}

public List<String> getSuggestions(String misspelledWord)

{

@SuppressWarnings("unchecked")

List<Word> su99esti0ns = spellChecker.getSuggestions(misspelledWord, 0);

List<String> suggestions = new ArrayList<String>();

for (Word suggestion : su99esti0ns)

{

suggestions.add(suggestion.getWord());

}

return suggestions;

}

@Override

public void spellingError(SpellCheckEvent event)

{

event.ignoreWord(true);

misspelledWords.add(event.getInvalidWord());

}

public static void main(String[] args)

{

}

}

**Software Version:**  Version 1.0

**Evaluation:**

**Dataset:**

The dataset for the system is a JPEG image which has one word in Calibri font in black color. A sample of the input and the corresponding output are given below:

**Case 1:**

**Input:**



**Output:**



**Case 2:**

**Input:**



**Output:**

