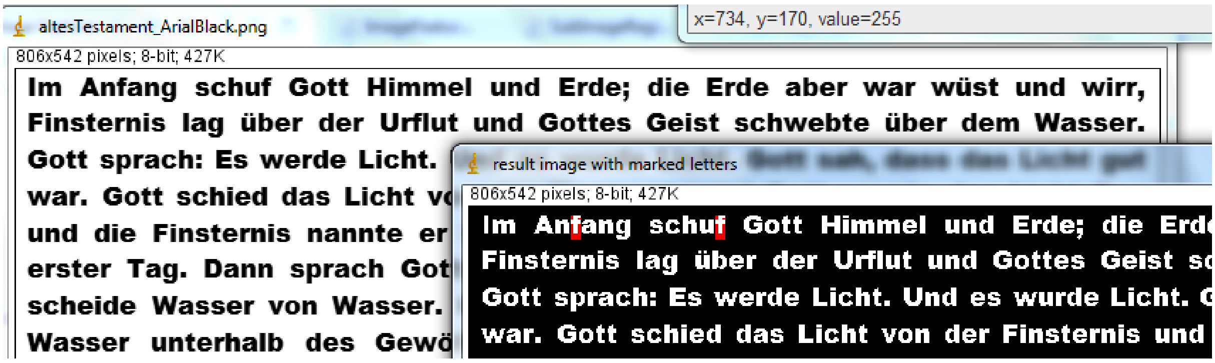
BVA2 – **Exercise 06:** ***Pattern Recognition - OCR***

[18 points]

Application of OCR pattern recognition to mark target letter in a text

Implement an ImageJ plugin to mark all occurrences of a target letter within an input text provided as grayscale image as illustrated below:


altesTestament_ArialBlack.png
x=734, y=170, value=255
806x542 pixels; 8-bit; 427K
result image with marked letters
806x542 pixels; 8-bit; 427K

Im Anfang schuf Gott Himmel und Erde; die Erde aber war wüst und wirr,
Finsternis lag über der Urflut und Gottes Geist schwebte über dem Wasser.
Gott sprach: Es werde Licht.
war. Gott schied das Licht vo
und die Finsternis nannte er
erster Tag. Dann sprach Got
scheide Wasser von Wasser.
Wasser unterhalb des Gewö

Im Anfang schuf Gott Himmel und Erde; die Erd
Finsternis lag über der Urflut und Gottes Geist sc
Gott sprach: Es werde Licht. Und es wurde Licht. G
war. Gott schied das Licht von der Finsternis und

Example with reference character set to row=1, column=5, i.e. “f”. Some/most occurrences of “f” are marked in red as final result.

1. Basic Implementation [14 points]

For implementation of this exercise utilized the provided template files and implement the defined interfaces. The following steps need to be implemented. Always evaluate all of the achieved intermediate results:

- At first the input image needs to be binarized utilizing a proper threshold value.
- Separate the input image horizontally by splitting into sub-images that are containing the letters, i.e. lines. Then further separate the lines by vertical splitting to separate into character/letter images. Therefore, the sub-image data structure (`class SubImageRegion`) needs to be utilized. To separate the characters, the method `splitCharacters` needs to be implemented utilizing “fire-through” strategy. As an alternative, common region growing could be applied too (*cons: no ordering of letters in rows and columns, consider problems related to “i”,…*).
  ```
  public Vector<Vector<SubImageRegion>> splitCharacters(int[][] inImg, int width, int height, int BG_val, int FG_val)
  public Vector<SubImageRegion> splitCharactersVertically(SubImageRegion rowImage, int BG_val, int FG_val, int[][] origImg)
  ```
- Define the reference character (user input) and calculate the feature vector: `double[] calcFeatureArr(SubImageRegion region, int FGval, Vector<ImageFeatureBase> featuresToUse)`
  All the utilized features are derived from abstract base class `abstract class ImageFeatureBase` and necessitate implementation of the evaluation function `public abstract double CalcFeatureVal(SubImageRegion imgRegion, int FG_val)`. Implement at least all of the pre-defined features.
- For normalization, the mean feature values need to be calculated utilizing all characters in the provided text image:
  ```
  double[] normArr = calculateNormArr(splittedChars, BG_VAL, featureVect);
  ```

- Finally, classify all characters that show some level of similarity compared to the provided reference character. For comparison of the feature vectors, utilize correlation coefficient calculation and normalization.
- All letters that show a correlation coefficient above a certain confidence level are marked in the final result image. To allow for good results due to parameter tuning, the correlation coefficient threshold should be defined/adapted by the user.

Extensively test your implementation and evaluate the classification accuracy.

- Which letters can be detected in a confident way and which letters lead to problems – and why?
- Are all fonts applicable to this kind of OCR strategy – why or why not?
- Does classification accuracy depend on the other characters in the specific line – if that's the case: why and how to overcome this issue?

2. Advanced Implementation [4 points]

- Implement at least 3 additional features that improve robustness of the basic implementation.
- Ensure that the split characters image region is shrinked to its bounding box. How can that help to improve result quality?
- Discuss the normalization process – how does character occurrence probability influence the results?
- Discuss how the classification process itself could be improved. Are there better strategies for feature-based classification?
- How does correlation of some of the features itself influence the achievable classification results (*e.g.* ***max-distance-from-centroid*** *will somehow be correlated to the specific* ***width***)?

1 Basic Implementation

1.1 Lösungsidee

- **splitCharacters(int[][] InImg, int width, int height, int BG_val, int FG_val):** In dieser Methode werden Zeilen erkannt. Solang eine Reihe rein aus BG_val besteht, wird diese nicht in das SubImage mitaufgenommen. Sobald nun eine Zeile mit mindestens einem FG_val kommt, wird diese als Startpunkt des SubImage markiert. Solang nun keine Reihe mit rein aus BG_val bestehenden Pixeln kommt, wird fortgefahren. Kommt nun eine Zeile mit lediglich BG_val, endet hier das SubImage und es kann im Vector gespeichert werden. Dieser Vorgang wird solange wiederholt, bis das Ende des Images erreicht ist.

- **splitCharactersVertically(int[][] InImg, int width, int height, int BG_val, int FG_val):** Diese Methode wird ähnlich zur vorherigen implementiert. Es wird lediglich das Durchlaufen des Bildes auf spaltenweise geändert.

- **calcFeatureArr(SubImageRegion region, int FGval, Vector<ImageFeatureBase> featuresToUse:** Hier wird für die angegebene SubImageRegion jeder Feature-Wert der angegebenen Features berechnet. Dies wird über die Methode *CalcFeatureVal* berechnet, die jedes Feature implementieren muss.

- **calculateNormArr(splittedChars, BG_VAL, featureVect):** In dieser Methode wird für jedes Feature der Mittelwert aller SubImages ausgerechnet. Dies dient zum Normalisieren, da sonst höhere Werte bevorzugt werden.

1.1.1 Features

- **F1: number of pixels:** Bei diesem Feature werden die Anzahl an FG_val-gefärbten Pixel im SubImage berechnet.

- **F2: extent in x-direction (max width):** Hier wird die maximale Anzahl an FG_val Pixel einer Reihe berechnet. Dazu wird eine Counter-Variable definiert.

- **F3: extent in y-direction (max height):** Hier wird die maximale Anzahl an FG_val Pixel einer Spalte berechnet. Dazu wird eine Counter-Variable definiert.

- **F4: average distance from centroid:** Zuerst wird die Berechnung des Zentroide in einer eigenen Hilfsfunktion implementiert, welche später auch für die restlichen Features (die den Zentroiden benötigen) verwendet werden kann. Die X-Koordinate des Zentroiden wird bestimmt, indem alle X-Koordinaten der Pixel, die zum jeweiligen Buchstaben gehören, aufsummiert werden und durch die Anzahl der beteiligten Pixel dividiert werden. Für die Y-Koordinate werden dann die jeweiligen y-Koordinaten aufsummiert. Um die durchschnittliche Distanz zum Zentroiden zu berechnen werden alle Distanzen zu allen Pixeln die zum Buchstaben gehören aufsummiert und durch die Anzahl der Pixel geteilit. Zum bestimmen der Distanz wird die Formel $\sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$ verwendet.

- **F5: min distance from centroid:** Um die minimale Distanz zum Zentroiden zu bestimmen wird der Zentroide wieder mit der zuvor implementierten Hilfsfunktion bestimmt, um dann die minimale Distanz zu finden.

- **F6: max distance from centroid:** Funktioniert gleich wie das Bestimmen der minimalen Distanz, nur das hier die maximale Distanz als Ergebnis gespeichert wird.

- **F7: circularity:** Zur Berechnung der "Circularity" wirde folgende Formel herangezogen: $circularity = \frac{4\pi * area}{perimeter^2}$. Wobei "area" die Anzahl der Pixel eines Zeichens und "perimeter" die

Anzahl der Pixel des Umfangs sind. Beide Werte können über die ImageJ-Klasse "Analyzer" ermittelt werden.

- **F8: rel pos centroid within bounding box, x-direction:** Zur Berechnung des Zentroiden kann wieder die Hilfsfunktion von vorher verwendet werden. Danach kann die relative Position in x-Richtung berechnet werden.

- **F9: rel pos centroid within bounding box, y-direction:** Funktioniert Äquivalent zu F8, nur das hier die Position in y-Richtung berechnet wird.

1.2 Fragen

- **Which letters can be detected in a confident way and which letters lead to problems - and why?**

- **Are all fonts applicable to this kind of OCR strategy - why or why not?:** Nein, diese Strategie ist nicht mit allen Schriften möglich. Bei z.B. Schreibschriften oder Schriften, wo durchgehende Übergänge zwischen den Buchstaben sind, ist die Separierung der einzelnen Buchstaben nicht möglich. Auch bei handgeschriebene Buchstaben, wo dieselben Zeichen unterschiedlich aussehen (durch Handschrift), ist die Erkennung nicht exakt möglich.

- **Does classification accuracy depend on the other characters in the specific line - if that's the case: why and how to overcome this issue?** Ja, da z.B. ein i, wenn in der Zeile ein g vorhanden ist, eine größere Bounding-Box aufweist, als ein i, wenn in der Zeile kein g vorhanden ist. Dies kann bei der Berechnung der Features Probleme bereiten, da das SubImage bei beiden i's nicht ident ist. Man könnte z.B. jedes SubImage selbst noch einmal „zuschneiden" und so die restlichen leeren Zeilen wegschneiden.

1.3 Ausarbeitung

Listing 1: OCRanalysis_.java

```java
import ij.IJ;
import ij.ImagePlus;
import ij.gui.GenericDialog;
import ij.gui.Roi;
import ij.measure.Measurements;
import ij.measure.ResultsTable;
import ij.plugin.filter.Analyzer;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;
import ij.process.ImageStatistics;

import java.awt.Point;
import java.awt.Rectangle;
import java.util.Random;
import java.util.Vector;

public class OCRanalysis_ implements PlugInFilter {

    private ImagePlus imp;
    private ImageProcessor ip;

    public int setup(String arg, ImagePlus imp) {
        if (arg.equals("about")) {
            showAbout();
            return DONE;
        }

        this.imp = imp;

        return DOES_8G + DOES_RGB + DOES_STACKS + SUPPORTS_MASKING;
    } //setup

    //-------- the defined features ----------------
    public static int F_FGcount = 0;
    public static int F_MaxDistX = 1;
    public static int F_MaxDistY = 2;
    public static int F_AvgDistanceCentroide = 3;
    public static int F_MaxDistanceCentroide = 4;
    public static int F_MinDistanceCentroide = 5;
    public static int F_Circularity = 6;
    public static int F_CentroideRelPosX = 7;
    public static int F_CentroideRelPosY = 8;
    //---------------------------------------------


    public void run(ImageProcessor ip) {
        this.ip = ip;

        Vector<ImageFeatureBase> featureVect = new Vector<ImageFeatureBase>();
        featureVect.add(new ImageFeatureF_FGcount());
        featureVect.add(new ImageFeatureF_MaxDistX());
        featureVect.add(new ImageFeatureF_MaxDistY());
        featureVect.add(new ImageFeatureF_MaxDistanceCentroide());
        featureVect.add(new ImageFeatureF_MinDistanceCentroide());
        featureVect.add(new ImageFeatureF_AvgDistanceCentroide());
        featureVect.add(new ImageFeatureF_Circularity());
        featureVect.add(new ImageFeatureF_CentroideRelPosX());
        featureVect.add(new ImageFeatureF_CentroideRelPosY());


        byte[] pixels = (byte[]) ip.getPixels();
        int width = ip.getWidth();
        int height = ip.getHeight();
        int[][] inDataArrInt = ImageJUtility.convertFrom1DByteArr(pixels, width, height);

        //(1) at first do some binarization
        int FG_VAL = 0;
        int BG_VAL = 255;
        int MARKER_VAL = 127;
        int thresholdVal = 0;//?;

        int[] binaryThreshTF = ImageTransformationFilter.GetBinaryThresholdTF(255, thresholdVal, MARKER_VAL, FG_VAL, BG_VAL);
        int[][] binaryImgArr = ImageTransformationFilter.GetTransformedImage(inDataArrInt, width, height, binaryThreshTF);

        ImageJUtility.showNewImage(binaryImgArr, width, height, "binary image at threh = " + thresholdVal);

        //(2) split the image according to fire-trough or multiple region growing
        Vector<Vector<SubImageRegion>> splittedCharacters = splitCharacters(binaryImgArr, width, height, BG_VAL, FG_VAL);

        // for reasons of testing, visualize some of the split characters
        Random random = new Random();
        int randomLine = random.nextInt(splittedCharacters.size());
        for (SubImageRegion subImageRegion : splittedCharacters.get(randomLine)) {
            subImageRegion.showImage();
        }

        //let the user specify the target character
        final int[] max = {0};
        splittedCharacters.stream().forEach(e -> {
            if (e.size() > max[0]) {
                max[0] = e.size();
            }
        });
        GenericDialog dialog = createDialog("tgtCharRow", splittedCharacters.size(), "tgtCharCol",
        ↪  max[0]);
        int tgtCharRow = (int) dialog.getNextNumber();
        int tgtCharCol = (int) dialog.getNextNumber();
        System.out.println("Chosen row: " + tgtCharRow);
        System.out.println("Chosen col: " + tgtCharCol);

        SubImageRegion charROI = splittedCharacters.get(tgtCharRow).get(tgtCharCol);
        ImageJUtility.showNewImage(charROI.subImgArr, charROI.width, charROI.height, "char at pos
        ↪  " + tgtCharRow + " / " + tgtCharCol);

        //calculate features of reference character
        double[] featureResArr = calcFeatureArr(charROI, FG_VAL, featureVect);
        printoutFeatureRes(featureResArr, featureVect);

        //calculate mean values for all features based on all characters
        //==> required for normalization
        double[] normArr = calculateNormArr(splittedCharacters, FG_VAL, featureVect);
        printoutFeatureRes(normArr, featureVect);

        int hitCount = 0; //count the number of detected characters


        //now detect all matching characters
        for (Vector<SubImageRegion> subImageRegionRow : splittedCharacters) {
            for (SubImageRegion subImageRegion : subImageRegionRow) {
                double[] currFeatureArr = calcFeatureArr(subImageRegion, FG_VAL, featureVect);
                if (isMatchingChar(currFeatureArr, featureResArr, normArr)) {
                    hitCount++;

                    binaryImgArr = markRegionInImage(binaryImgArr, subImageRegion, BG_VAL, BG_VAL
                    ↪  / 2);
                }
            }
        }

        IJ.log("# of letters detected = " + hitCount);

        ImageJUtility.showNewImage(binaryImgArr, width, height, "result image with marked
        ↪  letters");
    } //run

    public int[][] markRegionInImage(int[][] inImgArr, SubImageRegion imgRegion, int
    ↪  colorToReplace, int tgtColor) {
        for (int x = imgRegion.startX; x < imgRegion.startX + imgRegion.width; x++) {
            for (int y = imgRegion.startY; y < imgRegion.startY + imgRegion.height; y++) {
                if (inImgArr[x][y] == colorToReplace) {
                    inImgArr[x][y] = tgtColor;
                }
            }
        }

        return inImgArr;
    }

    boolean isMatchingChar(double[] currFeatureArr, double[] refFeatureArr, double[]
    ↪  normFeatureArr) {
        double CORR_COEFFICIENT_LIMIT = 0.5;

        double sxy = 0.0;
        double sx = 0.0;
        double sy = 0.0;
        for (int i = 0; i < normFeatureArr.length; i++) {
            double xi = Math.abs(currFeatureArr[i] - normFeatureArr[i]); // xi - x'
            double yi = Math.abs(refFeatureArr[i] - normFeatureArr[i]); // yi - y'
            sxy += xi * yi;
            sx += Math.pow(xi, 2);
            sy += Math.pow(yi, 2);

        }

        double correlCoeff = sxy / (Math.sqrt(sx) * Math.sqrt(sy));
        return correlCoeff > CORR_COEFFICIENT_LIMIT || correlCoeff < -CORR_COEFFICIENT_LIMIT;
    }


    void printoutFeatureRes(double[] featureResArr, Vector<ImageFeatureBase> featuresToUse) {
        IJ.log("========== features =========");
        for (int i = 0; i < featuresToUse.size(); i++) {
            IJ.log("res of F " + i + ", " + featuresToUse.get(i).description + " is " +
            ↪  featureResArr[i]);
        }
    }


    double[] calcFeatureArr(SubImageRegion region, int FGval, Vector<ImageFeatureBase>
    ↪  featuresToUse) {
        double[] featureResArr = new double[featuresToUse.size()];
        for (int i = 0; i < featuresToUse.size(); i++) {
            featureResArr[i] = featuresToUse.get(i).CalcFeatureVal(region, FGval);
        }

        return featureResArr;
    }

    double[] calculateNormArr(Vector<Vector<SubImageRegion>> inputRegions, int FGval,
    ↪  Vector<ImageFeatureBase> featuresToUse) {
        //calculate the average per feature to allow for normalization
        double[] returnArr = new double[featuresToUse.size()];
        for (int i = 0; i < featuresToUse.size(); i++) {
            double avg = 0.0;
            int count = 0;
            for (Vector<SubImageRegion> row : inputRegions) {
                for (SubImageRegion image : row) {
                    avg += featuresToUse.get(i).CalcFeatureVal(image, FGval);
                    count++;
                }
            }
            avg /= count;
            returnArr[i] = avg;
        }

        double min = Double.MAX_VALUE;
        double max = Double.MIN_VALUE;
        double mean = 0.0;
        for(int i = 0; i < returnArr.length; i++){
            if(returnArr[i] > max){
                max = returnArr[i];
            }
            if(returnArr[i] < min){
                min = returnArr[i];
            }
            mean += returnArr[i];
        }
        mean /= returnArr.length;

        for(int i = 0; i < returnArr.length; i++){
            returnArr[i] = (returnArr[i] - mean) / (max - min);
        }

        return returnArr;
    }

    //outer Vector ==> lines, inner vector characters per line, i.e. columns
    public Vector<Vector<SubImageRegion>> splitCharacters(int[][] inImg, int width, int height,
    ↪   int BG_val, int FG_val) {
        Vector<Vector<SubImageRegion>> returnCharMatrix = new Vector<Vector<SubImageRegion>>();

        int startY = 0;
        boolean foundFG = false;
        boolean foundOnlyBackgroundInLine = true;
        for (int y = 0; y < height; y++) {
            foundOnlyBackgroundInLine = true;
            for (int x = 0; x < width; x++) {
                // if the value is a FG_val set start points
                // go on until there is a completely white line
                if (inImg[x][y] == FG_val) {
                    if (!foundFG) {
                        startY = y;
                    }
                    foundFG = true;
                    foundOnlyBackgroundInLine = false;
                    break;
                }

            }
            // found a completely background line and there was a FG_val before
            // so this is a new region
            if (foundOnlyBackgroundInLine && foundFG) {
                foundFG = false;
                SubImageRegion subImageRegion = new SubImageRegion(0, startY, width, y - 1 -
                ↪   startY, inImg);
                Vector<SubImageRegion> horizontalRegions =
                ↪   splitCharactersVertically(subImageRegion, BG_val, FG_val, inImg);
                returnCharMatrix.add(horizontalRegions);
            }
        }

        return returnCharMatrix;
    }

    public Vector<SubImageRegion> splitCharactersVertically(SubImageRegion rowImage, int BG_val,
    ↪   int FG_val, int[][] origImg) {
        Vector<SubImageRegion> returnCharArr = new Vector<SubImageRegion>();

        int startX = 0;
        int startY = 0;
        boolean foundFG = false;
        boolean foundOnlyBackgroundInLine = true;
        for (int x = rowImage.startX; x < rowImage.width; x++) {
            foundOnlyBackgroundInLine = true;
            for (int y = 0; y < rowImage.height; y++) {
                if (rowImage.subImgArr[x][y] == FG_val) {
                    if (foundFG == false) {
                        startX = x;
                        startY = rowImage.startY;
                    }
                    foundFG = true;
                    foundOnlyBackgroundInLine = false;
                    break;
                }

            }
            if (foundOnlyBackgroundInLine && foundFG) {
                foundFG = false;
                SubImageRegion subImageRegion = new SubImageRegion(startX, startY, x - startX, rowImage.height, origImg);
                returnCharArr.add(subImageRegion);
            }
        }

        return returnCharArr;
    }

    void showAbout() {
        IJ.showMessage("About Template_...",
                "this is a RegionGrowing_ template\n");
    } //showAbout


    //the features to implement


    class ImageFeatureF_FGcount extends ImageFeatureBase {

        public ImageFeatureF_FGcount() {
            this.description = "Pixelanzahl";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            double count = 0;

            for (int x = 0; x < imgRegion.width; x++) {
                for (int y = 0; y < imgRegion.height; y++) {
                    if (imgRegion.subImgArr[x][y] == FG_val) {
                        count++;
                    }
                }
            }

            return count;
        }
    }

    class ImageFeatureF_MaxDistX extends ImageFeatureBase {

        public ImageFeatureF_MaxDistX() {
            this.description = "maximale Ausdehnung in X-Richtung";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            int maxNumberOfFGInLine = 0;
            int counter = 0;

            for (int y = 0; y < imgRegion.height; y++) {
                counter = 0;
```

```java
                for (int x = 0; x < imgRegion.width; x++) {
                    if (imgRegion.subImgArr[x][y] == FG_val) {
                        counter++;
                    }
                }
                if (counter > maxNumberOfFGInLine) maxNumberOfFGInLine = counter;
            }
            return maxNumberOfFGInLine;
        }

    }

    class ImageFeatureF_MaxDistY extends ImageFeatureBase {

        public ImageFeatureF_MaxDistY() {
            this.description = "maximale Ausdehnung in Y-Richtung";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            int maxNumberOfFGInLine = 0;
            int counter = 0;

            for (int x = 0; x < imgRegion.width; x++) {
                counter = 0;
                for (int y = 0; y < imgRegion.height; y++) {
                    if (imgRegion.subImgArr[x][y] == FG_val) {
                        counter++;
                    }
                }
                if (counter > maxNumberOfFGInLine) maxNumberOfFGInLine = counter;
            }
            return maxNumberOfFGInLine;
        }

    }


    private double calcDistance(int x1, int y1, int x2, int y2) {
        return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
    }

    private Point calcCentroid(SubImageRegion imgRegion, int FG_val) {
        // https://ask2mujahed.wordpress.com/category/research-topics/ocr/

        int centroidX = 0;
        int centroidY = 0;
        int cnt = 0;
        for (int x = 0; x < imgRegion.width; x++) {
            for (int y = 0; y < imgRegion.height; y++) {
                if (imgRegion.subImgArr[x][y] == FG_val) {
                    centroidX += x;
                    centroidY += y;
                    cnt++;
                }
            }
        }

        return new Point(centroidX / cnt, centroidY / cnt);
    }

    class ImageFeatureF_AvgDistanceCentroide extends ImageFeatureBase {

        public ImageFeatureF_AvgDistanceCentroide() {
            this.description = "mittlere Distanz zum Centroide";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            Point centroid = calcCentroid(imgRegion, FG_val);

            double avgDist = 0;
            int cnt = 0;
            for (int x = 0; x < imgRegion.width; x++) {
                for (int y = 0; y < imgRegion.height; y++) {
                    if (imgRegion.subImgArr[x][y] == FG_val) {
                        avgDist += calcDistance(centroid.x, centroid.y, x, y);
                        cnt++;
                    }
                }
            }

            return avgDist / cnt;
        }
    }

    class ImageFeatureF_MaxDistanceCentroide extends ImageFeatureBase {

        public ImageFeatureF_MaxDistanceCentroide() {
            this.description = "maximale Distanz zum Centroide";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            Point centroid = calcCentroid(imgRegion, FG_val);

            double maxDist = 0;
            for (int x = 0; x < imgRegion.width; x++) {
                for (int y = 0; y < imgRegion.height; y++) {
                    if (imgRegion.subImgArr[x][y] == FG_val) {
                        double actDist = calcDistance(centroid.x, centroid.y, x, y);
                        if (actDist > maxDist) {
                            maxDist = actDist;
                        }
                    }
                }
            }

            return maxDist;
        }
    }

    class ImageFeatureF_MinDistanceCentroide extends ImageFeatureBase {

        public ImageFeatureF_MinDistanceCentroide() {
            this.description = "minimale Distanz zum Centroide";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            Point centroid = calcCentroid(imgRegion, FG_val);

            double minDist = Double.MAX_VALUE;
            for (int x = 0; x < imgRegion.width; x++) {
                for (int y = 0; y < imgRegion.height; y++) {
                    if (imgRegion.subImgArr[x][y] == FG_val) {
                        double actDist = calcDistance(centroid.x, centroid.y, x, y);
                        if (actDist < minDist) {
                            minDist = actDist;
                        }
                    }
                }
            }

            return minDist;
        }

    }

    class ImageFeatureF_Circularity extends ImageFeatureBase {

        ImageStatistics stats;
        Analyzer analyzer;

        public ImageFeatureF_Circularity() {
            this.description = "Circularitaet";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            Roi roi = new Roi(imgRegion.startX, imgRegion.startY, imgRegion.width,
            ↪ imgRegion.height);
            ip.setRoi(roi);

            int measurements = Analyzer.getMeasurements();
            measurements |= Measurements.AREA + Measurements.PERIMETER;
            Analyzer.setMeasurements(measurements);
            analyzer = new Analyzer();
            stats = imp.getStatistics(measurements);

            analyzer.saveResults(stats, roi);
            ResultsTable rt = Analyzer.getResultsTable();
            int counter = rt.getCounter();
            double area = rt.getValueAsDouble(ResultsTable.AREA, counter - 1);
            double perimeter = rt.getValueAsDouble(ResultsTable.PERIMETER, counter - 1);

            return perimeter == 0.0 ? 0.0 : 4.0 * Math.PI * (area / (perimeter * perimeter));
        }

    }

    class ImageFeatureF_CentroideRelPosX extends ImageFeatureBase {

        public ImageFeatureF_CentroideRelPosX() {
            this.description = "relative x-Position des Centroide";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            Point centroid = calcCentroid(imgRegion, FG_val);
            return calcDistance(centroid.x, centroid.y, centroid.x, 0);
        }

    }

    class ImageFeatureF_CentroideRelPosY extends ImageFeatureBase {

        public ImageFeatureF_CentroideRelPosY() {
            this.description = "relative y-Position des Centroide";
        }

        public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
            Point centroid = calcCentroid(imgRegion, FG_val);
            return calcDistance(centroid.x, centroid.y, 0, centroid.y);
        }

    }

    private GenericDialog createDialog(String rowName, int maxRow, String colName, int maxCol) {
        GenericDialog gd = new GenericDialog("User Input");
        gd.addSlider(rowName, 0, maxRow - 1, 1);
        gd.addSlider(colName, 0, maxCol - 1, 1);
        gd.showDialog();
        if (gd.wasCanceled()) {
            return null;
        } //if
        return gd;
    }

} //class OCRanalysisTemplate
```

1.4 Tests