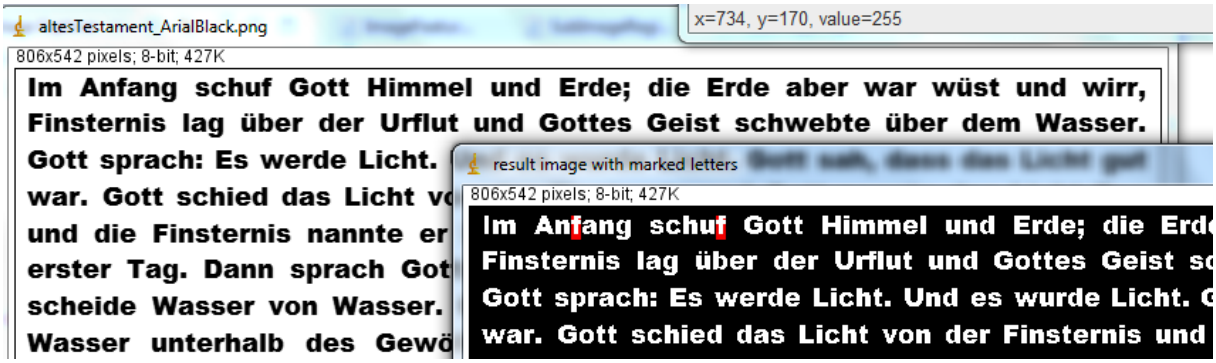
BVA2 – **Exercise 06:** ***Pattern Recognition - OCR***

[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:



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

1.0.1 Lösungsidee

TODO

1.0.2 Ausarbeitung

Listing 1: OCRanalysis_.java

```
1  import ij.IJ;
2  import ij.ImagePlus;
3  import ij.gui.GenericDialog;
4  import ij.gui.Roi;
5  import ij.measure.Measurements;
6  import ij.measure.ResultsTable;
7  import ij.plugin.filter.Analyzer;
8  import ij.plugin.filter.PlugInFilter;
9  import ij.process.ImageProcessor;
10 import ij.process.ImageStatistics;
11
12 import java.awt.Point;
13 import java.awt.Rectangle;
14 import java.util.Random;
15 import java.util.Vector;
16
17 public class OCRanalysis_ implements PlugInFilter {
18
19   private ImagePlus imp;
20   private ImageProcessor ip;
21
22     public int setup(String arg, ImagePlus imp) {
23         if (arg.equals("about")) {
24             showAbout();
25             return DONE;
26         }
27
28         this.imp = imp;
29
30         return DOES_8G + DOES_RGB + DOES_STACKS + SUPPORTS_MASKING;
31     } //setup
32
33     //-------- the defined features ----------------
34     public static int F_FGcount = 0;
35     public static int F_MaxDistX = 1;
36     public static int F_MaxDistY = 2;
37     public static int F_AvgDistanceCentroide = 3;
38     public static int F_MaxDistanceCentroide = 4;
39     public static int F_MinDistanceCentroide = 5;
40     public static int F_Circularity = 6;
41     public static int F_CentroideRelPosX = 7;
42     public static int F_CentroideRelPosY = 8;
43     //----------------------------------------------
44
45
46     public void run(ImageProcessor ip) {
47       this.ip = ip;
48
49         Vector<ImageFeatureBase> featureVect = new Vector<ImageFeatureBase>();
50         featureVect.add(new ImageFeatureF_FGcount());
51         featureVect.add(new ImageFeatureF_MaxDistX());
52         featureVect.add(new ImageFeatureF_MaxDistY());
```

```
53          featureVect.add(new ImageFeatureF_MaxDistanceCentroide());
54          featureVect.add(new ImageFeatureF_MinDistanceCentroide());
55          featureVect.add(new ImageFeatureF_AvgDistanceCentroide());
56          featureVect.add(new ImageFeatureF_Circularity());
57
58
59          byte[] pixels = (byte[]) ip.getPixels();
60          int width = ip.getWidth();
61          int height = ip.getHeight();
62          int[][] inDataArrInt = ImageJUtility.convertFrom1DByteArr(pixels, width, height);
63
64          //(1) at first do some binarization
65          int FG_VAL = 0;
66          int BG_VAL = 255;
67          int MARKER_VAL = 127;
68          int thresholdVal = 0;//?;
69
70          int[] binaryThreshTF = ImageTransformationFilter.GetBinaryThresholdTF(255, thresholdVal,
            ↪  MARKER_VAL, FG_VAL, BG_VAL);
71          int[][] binaryImgArr = ImageTransformationFilter.GetTransformedImage(inDataArrInt, width,
            ↪  height, binaryThreshTF);
72
73          ImageJUtility.showNewImage(binaryImgArr, width, height, "binary image at threh = " +
            ↪  thresholdVal);
74
75          //(2) split the image according to fire-trough or multiple region growing
76          Vector<Vector<SubImageRegion>> splittedCharacters = splitCharacters(binaryImgArr, width,
            ↪  height, BG_VAL, FG_VAL);
77
78          // for reasons of testing, visualize some of the split characters
79          Random random = new Random();
80          int randomLine = random.nextInt(splittedCharacters.size());
81          for (SubImageRegion subImageRegion : splittedCharacters.get(randomLine)) {
82              subImageRegion.showImage();
83          }
84
85          //let the user specify the target character
86          final int[] max = {0};
87          splittedCharacters.stream().forEach(e -> {
88              if (e.size() > max[0]) {
89                  max[0] = e.size();
90              }
91          });
92          GenericDialog dialog = createDialog("tgtCharRow", splittedCharacters.size(), "tgtCharCol",
            ↪  max[0]);
93          int tgtCharRow = (int) dialog.getNextNumber();
94          int tgtCharCol = (int) dialog.getNextNumber();
95          System.out.println("Chosen row: " + tgtCharRow);
96          System.out.println("Chosen col: " + tgtCharCol);
97
98          SubImageRegion charROI = splittedCharacters.get(tgtCharRow).get(tgtCharCol);
99          ImageJUtility.showNewImage(charROI.subImgArr, charROI.width, charROI.height, "char at pos
            ↪  " + tgtCharRow + " / " + tgtCharCol);
100
101         //calculate features of reference character
102         double[] featureResArr = calcFeatureArr(charROI, FG_VAL, featureVect);
103         printoutFeatureRes(featureResArr, featureVect);
104
105 //          //TODO calculate mean values for all features based on all characters
106 //          //==> required for normalization
107         double[] normArr = calculateNormArr(splittedCharacters, BG_VAL, featureVect);
108         printoutFeatureRes(normArr, featureVect);
109 //
```

```
110 //        int hitCount = 0; //count the number of detected characters
111 //
112 //        //TODO: now detect all matching characters
113 //        //forall SubImageRegion sir in splittedCharacters
114 //        //if isMatchingChar(..,sir,..) then markRegionInImage(..,sir,..)
115 //
116 //
117 //        IJ.log("# of letters detected = " + hitCount);
118 //
119 //        bva2.ImageJUtility.showNewImage(binaryImgArr, width, height, "result image with marked
    ↪  letters");
120
121     } //run
122
123     public int[][] markRegionInImage(int[][] inImgArr, SubImageRegion imgRegion, int
    ↪  colorToReplace, int tgtColor) {
124         //TODO: implementation required
125         return inImgArr;
126     }
127
128     boolean isMatchingChar(double[] currFeatureArr, double[] refFeatureArr, double[]
    ↪  normFeatureArr) {
129         double CORR_COEFFICIENT_LIMIT = -1;//?;
130
131         //TODO: implementation required
132
133
134         return false;
135     }
136
137
138     void printoutFeatureRes(double[] featureResArr, Vector<ImageFeatureBase> featuresToUse) {
139         IJ.log("========== features =========");
140         for (int i = 0; i < featuresToUse.size(); i++) {
141             IJ.log("res of F " + i + ", " + featuresToUse.get(i).description + " is " +
                ↪  featureResArr[i]);
142         }
143     }
144
145
146     double[] calcFeatureArr(SubImageRegion region, int FGval, Vector<ImageFeatureBase>
    ↪  featuresToUse) {
147         //TODO implementation required
148         double[] featureResArr = new double[featuresToUse.size()];
149         for (int i = 0; i < featuresToUse.size(); i++) {
150             featureResArr[i] = featuresToUse.get(i).CalcFeatureVal(region, FGval);
151         }
152
153         return featureResArr;
154     }
155
156     double[] calculateNormArr(Vector<Vector<SubImageRegion>> inputRegions, int FGval,
    ↪  Vector<ImageFeatureBase> featuresToUse) {
157         //calculate the average per feature to allow for normalization
158         double[] returnArr = new double[featuresToUse.size()];
159         //TODO implementation required
160
161         return returnArr;
162     }
163
164     //outer Vector ==> lines, inner vector characters per line, i.e. columns
165     public Vector<Vector<SubImageRegion>> splitCharacters(int[][] inImg, int width, int height,
    ↪  int BG_val, int FG_val) {
```

```
166         Vector<Vector<SubImageRegion>> returnCharMatrix = new Vector<Vector<SubImageRegion>>();
167
168         int startY = 0;
169         boolean foundFG = false;
170         boolean foundOnlyBackgroundInLine = true;
171         for (int y = 0; y < height; y++) {
172             foundOnlyBackgroundInLine = true;
173             for (int x = 0; x < width; x++) {
174                 // if the value is a FG_val set start points
175                 // go on until there is a completely white line
176                 if (inImg[x][y] == FG_val) {
177                     if (foundFG == false) {
178                         startY = y;
179                     }
180                     foundFG = true;
181                     foundOnlyBackgroundInLine = false;
182                     break;
183                 }
184
185             }
186             // found a completely background line and there was a FG_val before
187             // so this is a new region
188             if (foundOnlyBackgroundInLine && foundFG) {
189                 foundFG = false;
190                 SubImageRegion subImageRegion = new SubImageRegion(0, startY, width, y - 1 -
                    ↪ startY, inImg);
191                 Vector<SubImageRegion> horizontalRegions =
                    ↪ splitCharactersVertically(subImageRegion, BG_val, FG_val, inImg);
192                 returnCharMatrix.add(horizontalRegions);
193             }
194         }
195
196         return returnCharMatrix;
197     }
198
199     public Vector<SubImageRegion> splitCharactersVertically(SubImageRegion rowImage, int BG_val,
    ↪ int FG_val, int[][] origImg) {
200         Vector<SubImageRegion> returnCharArr = new Vector<SubImageRegion>();
201
202         int startX = 0;
203         int startY = 0;
204         boolean foundFG = false;
205         boolean foundOnlyBackgroundInLine = true;
206         for (int x = rowImage.startX; x < rowImage.width; x++) {
207             foundOnlyBackgroundInLine = true;
208             for (int y = 0; y < rowImage.height; y++) {
209                 if (rowImage.subImgArr[x][y] == FG_val) {
210                     if (foundFG == false) {
211                         startX = x;
212                         startY = rowImage.startY;
213                     }
214                     foundFG = true;
215                     foundOnlyBackgroundInLine = false;
216                     break;
217                 }
218
219             }
220             if (foundOnlyBackgroundInLine && foundFG) {
221                 foundFG = false;
222                 SubImageRegion subImageRegion = new SubImageRegion(startX, startY, x - startX,
                    ↪ rowImage.height, origImg);
223                 returnCharArr.add(subImageRegion);
224             }
```

```
225         }
226
227         return returnCharArr;
228     }
229
230     void showAbout() {
231         IJ.showMessage("About Template_...",
232                 "this is a RegionGrowing_ template\n");
233     } //showAbout
234
235
236     //the features to implement
237
238
239     class ImageFeatureF_FGcount extends ImageFeatureBase {
240
241         public ImageFeatureF_FGcount() {
242             this.description = "Pixelanzahl";
243         }
244
245         public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
246             double count = 0;
247
248             for (int x = 0; x < imgRegion.width; x++) {
249                 for (int y = 0; y < imgRegion.height; y++) {
250                     if (imgRegion.subImgArr[x][y] == FG_val) {
251                         count++;
252                     }
253                 }
254             }
255
256             return count;
257         }
258     }
259
260     class ImageFeatureF_MaxDistX extends ImageFeatureBase {
261
262         public ImageFeatureF_MaxDistX() {
263             this.description = "maximale Ausdehnung in X-Richtung";
264         }
265
266         public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
267             int maxNumberOfFGInLine = 0;
268             int counter = 0;
269
270             for (int y = 0; y < imgRegion.height; y++) {
271                 counter = 0;
272                 for (int x = 0; x < imgRegion.width; x++) {
273                     if (imgRegion.subImgArr[x][y] == FG_val) {
274                         counter++;
275                     }
276                 }
277                 if (counter > maxNumberOfFGInLine) maxNumberOfFGInLine = counter;
278             }
279             return maxNumberOfFGInLine;
280         }
281
282     }
283
284     class ImageFeatureF_MaxDistY extends ImageFeatureBase {
285
286         public ImageFeatureF_MaxDistY() {
287             this.description = "maximale Ausdehnung in Y-Richtung";
```

```
288         }
289
290         public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
291             int maxNumberOfFGInLine = 0;
292             int counter = 0;
293
294             for (int x = 0; x < imgRegion.width; x++) {
295                 counter = 0;
296                 for (int y = 0; y < imgRegion.height; y++) {
297                     if (imgRegion.subImgArr[x][y] == FG_val) {
298                         counter++;
299                     }
300                 }
301                 if (counter > maxNumberOfFGInLine) maxNumberOfFGInLine = counter;
302             }
303             return maxNumberOfFGInLine;
304         }
305
306     }
307
308
309     private double calcDistance(int x1, int y1, int x2, int y2) {
310       return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
311     }
312
313     private Point calcCentroid(SubImageRegion imgRegion, int FG_val) {
314       // https://ask2mujahed.wordpress.com/category/research-topics/ocr/
315
316       int centroidX = 0;
317       int centroidY = 0;
318       int cnt = 0;
319       for(int x = 0; x < imgRegion.width; x++) {
320         for(int y = 0; y < imgRegion.height; y++) {
321           if(imgRegion.subImgArr[x][y] == FG_val) {
322             centroidX += x;
323             centroidY += y;
324             cnt++;
325           }
326         }
327       }
328
329       return new Point(centroidX / cnt, centroidY / cnt);
330     }
331
332     class ImageFeatureF_AvgDistanceCentroide extends ImageFeatureBase {
333
334         public ImageFeatureF_AvgDistanceCentroide() {
335             this.description = "mittlere Distanz zum Centroide";
336         }
337
338         public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
339           Point centroid = calcCentroid(imgRegion, FG_val);
340
341           double avgDist = 0;
342           int cnt = 0;
343           for(int x = 0; x < imgRegion.width; x++) {
344             for(int y = 0; y < imgRegion.height; y++) {
345               if(imgRegion.subImgArr[x][y] == FG_val) {
346                 avgDist += calcDistance(centroid.x, centroid.y, x, y);
347                 cnt++;
348               }
349             }
350           }
```

```
351
352             return avgDist / cnt;
353         }
354     }
355
356     class ImageFeatureF_MaxDistanceCentroide extends ImageFeatureBase {
357
358         public ImageFeatureF_MaxDistanceCentroide() {
359             this.description = "maximale Distanz zum Centroide";
360         }
361
362         public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
363           Point centroid = calcCentroid(imgRegion, FG_val);
364
365           double maxDist = 0;
366           for(int x = 0; x < imgRegion.width; x++) {
367             for(int y = 0; y < imgRegion.height; y++) {
368               if(imgRegion.subImgArr[x][y] == FG_val) {
369                 double actDist = calcDistance(centroid.x, centroid.y, x, y);
370                   if(actDist > maxDist) {
371                     maxDist = actDist;
372                   }
373               }
374             }
375           }
376
377           return maxDist;
378         }
379     }
380
381     class ImageFeatureF_MinDistanceCentroide extends ImageFeatureBase {
382
383         public ImageFeatureF_MinDistanceCentroide() {
384             this.description = "minimale Distanz zum Centroide";
385         }
386
387         public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
388           Point centroid = calcCentroid(imgRegion, FG_val);
389
390           double minDist = Double.MAX_VALUE;
391           for(int x = 0; x < imgRegion.width; x++) {
392             for(int y = 0; y < imgRegion.height; y++) {
393               if(imgRegion.subImgArr[x][y] == FG_val) {
394                 double actDist = calcDistance(centroid.x, centroid.y, x, y);
395                 if(actDist < minDist) {
396                   minDist = actDist;
397                 }
398               }
399             }
400           }
401
402             return minDist;
403         }
404
405     }
406
407     class ImageFeatureF_Circularity extends ImageFeatureBase {
408
409       ImageStatistics stats;
410       Analyzer analyzer;
411
412         public ImageFeatureF_Circularity() {
413             this.description = "Circularitaet";
```

```
414         }
415
416     public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
417           Roi roi = new Roi(imgRegion.startX, imgRegion.startY, imgRegion.width,
              ↪  imgRegion.height);
418         ip.setRoi(roi);
419
420         int measurements = Analyzer.getMeasurements();
421         measurements |= Measurements.AREA + Measurements.PERIMETER;
422         Analyzer.setMeasurements(measurements);
423         analyzer = new Analyzer();
424         stats = imp.getStatistics(measurements);
425
426         analyzer.saveResults(stats, roi);
427         ResultsTable rt =Analyzer.getResultsTable();
428         int counter = rt.getCounter();
429         double area = rt.getValueAsDouble(ResultsTable.AREA, counter-1);
430         double perimeter = rt.getValueAsDouble(ResultsTable.PERIMETER, counter-1);
431
432         return perimeter == 0.0 ? 0.0 : 4.0*Math.PI*(area/(perimeter*perimeter));
433         }
434
435     }
436
437     class ImageFeatureF_CentroideRelPosX extends ImageFeatureBase {
438
439         public ImageFeatureF_CentroideRelPosX() {
440             this.description = "relative x-Position des Centroide";
441         }
442
443         public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
444           Point centroid = calcCentroid(imgRegion, FG_val);
445             return calcDistance(centroid.x, centroid.y, centroid.x, 0);
446         }
447
448     }
449
450     class ImageFeatureF_CentroideRelPosY extends ImageFeatureBase {
451
452         public ImageFeatureF_CentroideRelPosY() {
453             this.description = "relative y-Position des Centroide";
454         }
455
456         public double CalcFeatureVal(SubImageRegion imgRegion, int FG_val) {
457           Point centroid = calcCentroid(imgRegion, FG_val);
458             return calcDistance(centroid.x, centroid.y, 0, centroid.y);
459         }
460
461     }
462
463     private GenericDialog createDialog(String rowName, int maxRow, String colName, int maxCol) {
464         GenericDialog gd = new GenericDialog("User Input");
465         gd.addSlider(rowName, 0, maxRow - 1, 1);
466         gd.addSlider(colName, 0, maxCol - 1, 1);
467         gd.showDialog();
468         if (gd.wasCanceled()) {
469             return null;
470         } //if
471         return gd;
472     }
473
474 } //class OCRanalysisTemplate
```

1.0.3 Tests