



UNIVERSITY
OF APPLIED SCIENCES
UPPER AUSTRIA

FH HAGENBERG

Signal- und Bildbearbeitung Uebung 02

Julian Buchgeher/Julian Kohr

S2210745004/S2210745012

Übungsaufgaben II, SVB1

11.11.2022

[Ziel: ca. 23 Punkte]

Aufgabe 2.1 Resampling und Interpolation [15]

- a) [6] Implementieren Sie einen Resampling Filter, der die Größenveränderung eines Eingangsbildes erlaubt. Der Gleitkomma-Skalierungsfaktor ist dabei vom Benutzer einzugeben und soll eine Vergrößerung / Verkleinerung um max. Faktor 10.0 erlauben. Die Interpolation ist dabei als *Nearest Neighbour* zu realisieren. Retournieren Sie das Ausgangsbild in einem neuen Fenster. Diskutieren Sie Ihre gewählte Berechnungsstrategie für die Koordinaten-Transformation (vgl. Folie 13-15).
- b) [6] Implementieren Sie zusätzlich eine Bi-Lineare Interpolation zum Resampling des Eingangsbildes. Retournieren Sie auch diesen Output in einem neuen Fenster. Berechnen Sie ferner ein Differenzbild und zeigen Sie dieses an. Wodurch entstehen die Unterschiede in den skalaren Pixelwerten?
- c) [3] Implementieren Sie eine Checker-Board Darstellung und vergleichen Sie die beiden Interpolationsstrategien. Wie würden Sie die beiden Interpolationsstrategien in Bezug auf Laufzeit und erzielbare Qualität charakterisieren?

Aufgabe 2.2 Klassifizierung mittels Kompression[15]

Die Übung kann „manuell“ durchgeführt werden. Für die statistische auswertung kann dabei z.B. „Excel“ oder jedes andere Statistikprogramm Ihrer Wahl eingesetzt werden.

- a) [15] Klassifizieren Sie Sprachen (textuelle Repräsentation) mittels Kompression. Wählen Sie dabei zumindest $n=8$ Klassen (d.h. Sprachen), für die Sie repräsentative Vergleichsdatensätze aufbereiten. Welche Vorverarbeitungen sind beim Aufbau der Vergleichsdatensätze bzw. Testdatensätze sinnvoll bzw. notwendig? Klassifizieren Sie dann Beispieltexte und werten Sie die Ergebnisse statistisch aus (exakte Treffer bzw. Rang, ev. Verwendung einer Konfusions-Matrix). Können die erzielten Ergebnisse bei vorliegender Testanzahl als statistisch signifikant betrachtet werden? Wie hängt die erzielbare Klassifikationsgenauigkeit mit der Anzahl der zu diskriminierenden Klassen zusammen?
- b) **OPTIONAL – nur für Interessierte/Expert*innen:** [20] Klassifizieren Sie Beispielbilder (*8bit Grauwert*) mittels Kompression. Wählen Sie zumindest $n=4$ Klassen, für die Sie repräsentative Vergleichsdatensätze aufbereiten. Welche Vorverarbeitungen sind beim Aufbau der Vergleichsdatensätze bzw. Testdatensätze sinnvoll bzw. notwendig? Klassifizieren Sie dann Beispieltexte und werten Sie die Ergebnisse statistisch aus (exakte Treffer bzw. Rang). Können die erzielten Ergebnisse bei vorliegender Testanzahl als statistisch signifikant betrachtet werden? Wie hängt die Klassifikationsgenauigkeit mit der Anzahl der zu diskriminierenden Klassen zusammen?

Aufgabe 2.3 Kompression und Code-Transformation[25]

Die nachfolgenden Beispiele können auch „händisch“ auf einem Blatt Papier erarbeitet und dann via Scan digitalisiert werden. Bitte auch die jeweiligen „Zusatzfragen“ zur Kompressionsrate bzw. Abhängigkeit von Homogenität bzw. Entropie beachten.

- a) [5] Komprimieren Sie die Sequenz **dabbababbbbbaaaabababccdd** ($n=4$ Symbole: $\{a,b,c,d\}$) händisch mittels Lempel-Ziv Kompression. Berechnen Sie die erzielbare Kompressionsrate.
- b) [8] Transformieren Sie die Sequenz **dabbabababbbbbaaaabababccdd** ($n=4$ Symbole: $\{a,b,c,d\}$) händisch mittels Huffmann-Coding in eine komprimierte Darstellung und geben Sie dabei auch den Huffmann-Baum an. Berechnen Sie die erzielbare Kompressionsrate bzw. die mittlere Codewort Länge.
Bei welcher eigenständig gewählten 10-stelligen Sequenz ist die Kompressionsrate maximal bzw. bei welcher 10-stelligen Sequenz ist die Kompressionsrate minimal? Von welcher Beschaffenheit der Daten hängt die erzielbare Kompressionsrate ab?
- c) [6] Komprimieren Sie händisch die Sequenz **11110101011110000011110001010** (30 Stellen, $n=2$ Symbole: $\{0,1\}$) mittels Runlength Coding. Berechnen Sie die erzielbare Kompressionsrate. Wie kann das Runlength Coding auf eine Symbolmenge $n>2$ erweitert werden und was ist dabei zu beachten? Demonstrieren Sie Ihre Erweiterung an einem selbst gewählten BSP und berechnen Sie die erzielbare Kompressionsrate. Von welcher Beschaffenheit der Daten hängt die erzielbare Kompressionsrate ab?
- d) [6] Berechnen Sie die Entropie der 30-stelligen Sequenz **221111226611122333345645112111** ($n=6$ Symbole: $\{1,2,3,4,5,6\}$). Bei welcher 10-stelligen Sequenz ist die Entropie maximal bzw. bei welcher 10-stelligen Sequenz ist die Entropie minimal? Welche Auswirkungen hat die Entropie in Bezug auf die erzielbare Kompression? Ist für die erzielbare Kompressionsrate dabei immer lediglich die Auftrittswahrscheinlichkeit entscheidend?

Inhaltsverzeichnis

1	2.1 Resampling und Interpolation	5
1.1	a	5
1.1.1	Implementierung	5
1.1.2	Berechnungsstrategie der Koordinatentransformation	9
1.2	b	9
1.3	c	11
2	2.2 Klassifizierung mittels Kompression	12
2.1	a	12
2.2	b	13
3	2.3 Kompression und Code-Transformation	16
3.1	a	16
3.2	b	17
3.3	c	18
3.4	d	18

1 2.1 Resampling und Interpolation

1.1 a

1.1.1 Implementierung

```

1 public class Resample_ implements PlugInFilter {
2
3     public int setup(String arg, ImagePlus imp) {
4         if (arg.equals("about"))
5             {showAbout(); return DONE;}
6         return DOES_8G;
7     } //setup
8
9     public static double[][] getResampledImage(double[][] inImg, int width, int height,
10        int tgtWidth, int tgtHeight, boolean useBilinearInterpolation) {
11
12        //initiate array for return image
13        double[][] returnImg = new double[tgtWidth][tgtHeight];
14
15        //this is strategy A)
16        //double scaleFactorW = tgtWidth / ((double)width);
17        //double scaleFactorH = tgtHeight / ((double)height);
18
19        //this is strategy B)
20        //calculate scale factor for width and height as of operation on slide 15 in the
21        //script
22        double scaleFactorW = (tgtWidth - 1.0) / (width - 1.0);
23        double scaleFactorH = (tgtHeight - 1.0) / (height - 1.0);
24
25        System.out.println("scale X = " + scaleFactorW + "scale Y = " + scaleFactorH);
26
27        //move all pixels FORWARD MAPPING from A ==> B
28        /*for(int x = 0; x < width; x++) {
29            for(int y = 0; y < height; y++) {
30                double posX = x * scaleFactorW;
31                double posY = y * scaleFactorH;
32                //example, pos I(2,3) with scale factor 4.0 ==> I'(8,12)
33                returnImg[(int)Math.round(posX)][(int)Math.round(posY)] =
34                    getNNinterpolatedValue(inImg, width, height, x, y)
35
36            } //for y
37        } //for x */
38
39        //get all resulting pixel in B from any position in A, from B ==> A, BACKWARD
40        //MAPPING
41        for(int x = 0; x < tgtWidth; x++) {
42            for(int y = 0; y < tgtHeight; y++) {
43                double posX = x / scaleFactorW;
44                double posY = y / scaleFactorH;
45                if(useBilinearInterpolation){
46                    returnImg[x][y] = getBilinearinterpolatedValue(inImg, width, height,
47                        posX, posY);
48                }
49                else {
50                    returnImg[x][y] = getNNinterpolatedValue(inImg, width, height, posX,
51                        posY);
52                }
53            } //for y
54        } //for x
55
56        //return array of return image
57        return returnImg;
58    }
59
60    public static double getNNinterpolatedValue(double[][] inImg, int width, int height,
61        double posX, double posY) {
62        //e.g. get value of inImg at position (posX, posY)(2.2, 3.7) ==> inImg[2][4]
63
64        //mechanism to round values to the nearest integer without Math.round
65        int posXint = (int)(posX + 0.5);
66        int posYint = (int)(posY + 0.5);
67
68    }
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
637
638
639
639
640
641
642
643
644
645
645
646
647
647
648
649
649
650
651
652
653
653
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
```

```

61     //let's check the range
62     if (posXint < 0)
63         posXint = 0;
64     if ( posYint < 0)
65         posYint = 0;
66     if (posXint >= width)
67         posXint = width - 1;
68     if ( posYint >= height)
69         posYint = height - 1;
70
71     //return value from input image that got calculated for output image
72     return inImg[posXint][posYint];
73 }
74
75 public static double getBilinearinterpolatedValue(double[][] inImg, int width, int
76 height, double posX, double posY) {
77
78     //rounding down positions
79     int posXint_0 = (int) posX;
80     int posYint_0 = (int) posY;
81
82     //get offset positions
83     int posXint_1 = posXint_0 + 1;
84     int posYint_1 = posYint_0 + 1;
85
86     //prevent getting out of bound
87     if (posXint_1 >= width)
88         posXint_1 = width - 1;
89     if (posYint_1 >= height)
90         posYint_1 = height - 1;
91
92     //get position delta as weight for value delta
93     double posXdelta = posX - posXint_0;
94     double posYdelta = posY - posYint_0;
95
96     //getting values of surrounding pixels
97     int value_0_0 = (int) inImg[posXint_0][posYint_0];
98     int value_1_0 = (int) inImg[posXint_1][posYint_0];
99     int value_0_1 = (int) inImg[posXint_0][posYint_1];
100    int value_1_1 = (int) inImg[posXint_1][posYint_1];
101
102    //get bilinear interpolated value as seen on slide 9 from script
103    int valueBIint = (int) Math.round(((1 - posXdelta) * (1 - posYdelta) * value_0_0
104        ) + (posXdelta * (1 - posYdelta) * value_1_0) + ((1 - posXdelta) * posYdelta
105        * value_0_1) + (posXdelta * posYdelta * value_1_1));
106
107    //return value from input image that got calculated for output image
108    return valueBIint;
109 }
110
111 public static double[][] getDifference(double[][] NN, double[][] BI, int tgtWidth,
112 int tgtHeight) {
113
114     //initiate array for return image
115     double[][] returnImg = new double[tgtWidth][tgtHeight];
116
117     //get all resulting delta pixel values by subtracting bilinear interpolated and
118     //nearest neighbor interpolated value
119     for(int x = 0; x < tgtWidth; x++) {
120         for(int y = 0; y < tgtHeight; y++) {
121             returnImg[x][y] = NN[x][y] - BI[x][y];
122
123             //if the value is negative add the positive value to it twice
124             if(returnImg[x][y] < 0){
125                 returnImg[x][y] += (returnImg[x][y] * (-1)) * 2;
126             } //if
127         } //for y
128     } //for x
129
130     //return array of return image
131     return returnImg;
132 }
133

```

```

129     public static double[][] getCheckerboard(double[][] NN, double[][] BI, int tgtWidth,
130         int tgtHeight) {
131
132     //initiate array for return image
133     double[][] returnImg = new double[tgtWidth][tgtHeight];
134
135     //generate a 2x2 checkerboard with nearest neighbor values in quadrant 1 and 4
136     for(int x = 0; x < tgtWidth; x++) {
137         for(int y = 0; y < tgtHeight; y++) {
138
139             //for all pixels placed inside the first and fourth quadrant use values
140             //of nearest neighbor interpolation
141             if(((x < tgtWidth / 2) && (y < tgtHeight / 2)) || ((x > tgtWidth / 2) &&
142                 (y > tgtHeight / 2))) {
143                 returnImg[x][y] = NN[x][y];
144             } //if
145
146             //else use values of bilinear interpolation
147             else {
148                 returnImg[x][y] = BI[x][y];
149             } //else
150         } //for y
151     } //for x
152
153     //return array of return image
154     return returnImg;
155 }
156
157     public void run(ImageProcessor ip) {
158
159         //initiate image data in arrays
160         byte[] pixels = (byte[])ip.getPixels();
161         int width = ip.getWidth();
162         int height = ip.getHeight();
163         int[] inDataArrInt = ImageJUtility.convertFrom1DByteArr(pixels, width, height)
164         ;
165         double[][] inDataArrDbl = ImageJUtility.convert.ToDoubleArr2D(inDataArrInt, width
166             , height);
167
168         //set default values for dialog variables
169         double scaleFactor = 2.11;
170         boolean useBilinearInterpolation = false;
171         boolean showDifference = false;
172         boolean showCheckerboard = false;
173
174         //generate dialog with input field for scale factor and checkbox for usage of
175         //bilinear interpolation
176         GenericDialog gd = new GenericDialog("resample");
177         gd.addNumericField("scale factor (between 0.1 & 10)", scaleFactor, 4);
178         gd.addCheckbox("use bilinear interpolation", useBilinearInterpolation);
179         gd.addCheckbox("show difference between NN and BI", showDifference);
180         gd.addCheckbox("show checkerboard (only available with show difference activated
181             )", showCheckerboard);
182         gd.showDialog();
183
184         //set user input for scale factor as variable sF
185         double sF = gd.getNextNumber();
186
187         //set scale factor to 10 for both, up and down scaling
188         if(sF <= 10 && sF >= 0.1) {
189             if(!gd.wasCanceled()) {
190                 scaleFactor = sF;
191                 useBilinearInterpolation = gd.getNextBoolean();
192                 showDifference = gd.getNextBoolean();
193                 showCheckerboard = gd.getNextBoolean();
194             } //if
195
196             else {
197                 return;
198             } //else
199         } //if
200
201         //if user input exceeds range for scale factor, show text dialog

```

```

195     else {
196         GenericDialog wn = new GenericDialog("scale factor out of range");
197         wn.addMessage("scale factor out of range");
198         wn.showDialog();
199         return;
200     } //else
201
202     //calculate and round target size based on the scale factor
203     //two different rounding methods:
204     double tgtWidth = (int)Math.round(width * scaleFactor); //either this
205     double tgtHeight = (int)(height * scaleFactor + 0.5); //or that
206
207     if(showDifference){
208         //generate resampled images in NN and BI + show difference between them
209         double[][] resampledImageNN = getResampledImage(inDataArrDbl, width, height,
210             (int)tgtWidth, (int)tgtHeight, false);
211         double[][] resampledImageBI = getResampledImage(inDataArrDbl, width, height,
212             (int)tgtWidth, (int)tgtHeight, true);
213         double[][] difference = getDifference(resampledImageNN, resampledImageBI, (
214             int)tgtWidth, (int)tgtHeight);
215
216         //show images
217         ImageJUtility.showNewImage(resampledImageNN, (int)tgtWidth, (int)tgtHeight,
218             "NN resized with factor " + scaleFactor);
219         ImageJUtility.showNewImage(resampledImageBI, (int)tgtWidth, (int)tgtHeight,
220             "BI resized with factor " + scaleFactor);
221         ImageJUtility.showNewImage(difference, (int)tgtWidth, (int)tgtHeight, "
222             difference");
223
224         if(showCheckerboard){
225             //generate checkerboard image with nearest neighbor in quadrant 1 and 4
226             double[][] checkerboard = getCheckerboard(resampledImageNN,
227                 resampledImageBI, (int)tgtWidth, (int)tgtHeight);
228
229             //show image
230             ImageJUtility.showNewImage(checkerboard, (int)tgtWidth, (int)tgtHeight,
231                 "checkerboard");
232         }
233     } //if
234
235     else {
236         //write resampled image into array
237         double[][] resampledImage = getResampledImage(inDataArrDbl, width, height, (
238             int) tgtWidth, (int) tgtHeight, useBilinearInterpolation);
239
240         //show image
241         if(useBilinearInterpolation) {
242             ImageJUtility.showNewImage(resampledImage, (int) tgtWidth, (int)
243                 tgtHeight, "BI resized with factor " + scaleFactor);
244         } //if
245         else {
246             ImageJUtility.showNewImage(resampledImage, (int) tgtWidth, (int)
247                 tgtHeight, "NN resized with factor " + scaleFactor);
248         } //else
249     } //else
250 } //run
251
252 void showAbout() {
253     IJ.showMessage("About Resample_...",
254         "this is a PluginFilter to resize input images\n");
255 } //showAbout
256
257 } //class Resample_

```

1.1.2 Berechnungsstrategie der Koordinatentransformation

Das Programm wurde mit mehreren Bildern getestet, jedoch haben sowohl die Boote, als auch der Clown oder Lena dieselben Ergebnisse geliefert. Die Ergebnisse werden weiter unten analysiert. Als Berechnungsstrategie für die Koordinatentransformation wurde der adaptierte Skalierungsfaktor verwendet. Dieser hat den Vorteil gegenüber der direkten Umrechnung, dass auch bei den Zielkoordinaten eine Übereinstimmung erzwungen wird. Nachteile sind allerdings, dass Werte am Beginn und Ende eine geringere Gewichtung haben. Zudem wird die Pixelgröße nicht berücksichtigt. Eine alternative Umrechnungsart, die diese Nachteile behebt, wäre die volle Überlappung der Pixelrepräsentierungen durch Berechnung der relativen Position. Wir haben immer einen Offset von einem halben Pixel, somit steht der Wert immer in der Mitte des Pixels.

1.2 b



Abbildung 1: Nearest Neighbour Interpolation, Faktor 3.0



Abbildung 2: Bilineare Interpolation, Faktor 3.0



Abbildung 3: Differenzbild

Die Unterschiede der skalaren Pixelwerte entstehen durch die Art und Weise, wie die Werte der Pixel im skalierten Bild durch die Methoden Nearest Neighbour und Bilinear Interpolation berechnet werden.

Bei der Nearest Neighbour Interpolation (NN) werden nur die Koordinaten betrachtet und der nächstgelegene Pixel durch Rundung ermittelt. Dessen Wert wird dann in den Zielpixel übernommen. Wenn ein Pixel und dessen benachbarter Pixel einen hohen Unterschied in ihren Werten haben, schlägt sich das auch im skalierten Bild nieder, da ein Teil der neuen Pixel den Wert des ersten und ein anderer Teil den Wert des zweiten Pixels erhält. Dazwischen werden keine neuen Werte generiert, was zu Stufenbildung führen kann.

Anders ist dies bei der Bilineararen Interpolation (BI) diese generiert für jeden Pixel einen Wert aus vier benachbarten Pixeln. Anhand der Zielkoordinate wird eine Gewichtung erstellt, mit der ein Mittelwert berechnet wird. Diese Operation muss drei Mal ausgeführt werden, was zu einer höheren Laufzeit führt.

1.3 c

Abbildung 4: Checkerboard, 1. u. 4. Quad: NN, 2. u. 3. Quad: BI

Die NN Interpolation führt zu Stufenbildung und Kanten, hat aber einen Vorteil durch die niedrige Laufzeit. Wenn man Bilder mit starken Kontrasten und Kanten skalieren möchte und/oder auf eine niedrige Laufzeit angewiesen ist, stellt die NN Interpolation eine gute Option dar.

Die Bilineare Interpolation besitzt eine höhere Laufzeit, da sie für jeden Zielpixel drei Operationen ausführen muss. Bei der NN Interpolation ist es nur eine, da lediglich die Koordinaten des Pixels, der den Quellwert liefert ermittelt werden muss. Die Bilineare Interpolation stellt Kanten im Zielbild auch nicht so gut dar, hat aber seine Vorteile bei homogenen Verläufen. Sie stellt aber einen guten Mittelweg zwischen NN und Kubischer Interpolation, die eine sehr hohe Laufzeit besitzt, dar.

Allgemein kann man sagen, dass die NN Interpolation zu einem Effekt führt, den wir als verpixelt bezeichnen würden und die Bilineare Interpolation eher zu einem verschwommenen Ergebnis.

2 2.2 Klassifizierung mittels Kompression

2.1 a

Für die Erkennung der textuellen Repräsentation (Sprachen) wurde von einer Website ein sogenannter Blindtext auf Deutsch verwendet. Dieser Text wurde anschließend auf 8 verschiedene Sprachen (mittels Deepl) übersetzt. Im Anschluss wurde überprüft, ob die Dateien annähernd die gleiche Datengröße besitzen. Neben den Haupttexten zum Vergleich wurden auf ähnliche Weise die Beispieltexte erstellt, wobei diese etwas kürzer waren als die Vergleichstexte.

Nach dieser Vorverarbeitung wurde zur besseren Struktur für jede Klassifizierung ein eigener Ordner angelegt. Dort wurde dann jeweils der Beispieltext in die Vergleichstexte kopiert und anschließend komprimiert. Dann wurden die Datengrößen der Zip-Ordner in die Tabelle neben der Größe vom komprimierten Vergleichstext eingetragen.

	A	B	C	D
1	Sprache	Datei	Datei + Beispieltext	Differenz
2	Deutsch	1,09 kB	1,43 kB	0,34 kB
3	Englisch	0,99 kB	1,36 kB	0,37 kB
4	Finnisch	1,03 kB	1,41 kB	0,38 kB
5	Französisch	1,11 kB	1,49 kB	0,38 kB
6	Japanisch	0,75 kB	1,19 kB	0,44 kB
7	Russisch	1,29 kB	1,75 kB	0,46 kB
8	Spanisch	1,05 kB	1,43 kB	0,38 kB
9	Türkisch	1,04 kB	1,42 kB	0,38 kB
10				
11				

Abbildung 5: Beispiel einer Klassifizierung

Nach dem Bilden der Differenz ist die kleinste Differenz als erkannte Sprache anzusehen. Diese wurde in der Konfusionsmatrix eingetragen. Die Excel-Tabelle mit der gesamten Auswertung wird angehängt, um bei Interesse die Daten vergleichen zu können.

		Ergebnis							
		Deutsch	Englisch	Finnisch	Französisch	Japanisch	Russisch	Spanisch	Türkisch
G e t e s t e t t	Sprachen	1	0	0	0	0	0	0	0
	Deutsch	1	0	0	0	0	0	0	0
	Englisch	0	1	0	0	0	0	0	0
	Finnisch	0	0	1	0	0	0	0	0
	Französisch	0	0	0	1	0	0	0	0
	Japanisch	0	0	0	0	1	0	0	0
	Russisch	0	0	0	0	0	1	0	0
	Spanisch	0	0	0	0	0	0	1	0
	Türkisch	0	0	0	0	0	0	0	1

Abbildung 6: Konfusionsmatrix der Sprachen

Aus der Konfusionsmatrix lässt sich sehr gut ablesen, dass in jedem Fall die Klassifizierung sehr gut funktioniert hat und immer die richtige Sprache erkannt wurde. Daraus kann man schließen, dass die Vorverarbeitung der Daten ausreichend war. Auf eine Platzierung der einzelnen Sprachen wurde verzichtet, wird aber anschließend bei den Bildern nachgeholt, da es dort mehr Sinn macht, weil dort nicht immer das richtige Bild erkannt wurde.

2.2 b

Für die Klassifizierung mit 8 Bit Graustufenbildern musste um einiges mehr Vorarbeit geleistet werden. Zuerst wurden passende Beispiele gesucht die mittels Screenshot zur Erzeugung eines PNG-Bildes ausgeschnitten wurden. Anschließend wurden diese Farbbilder in 8 Bit Graustufenbilder umgewandelt. Davor wurden die Bilder noch professionell mit Paint auf die gleiche Pixelgröße gebracht.

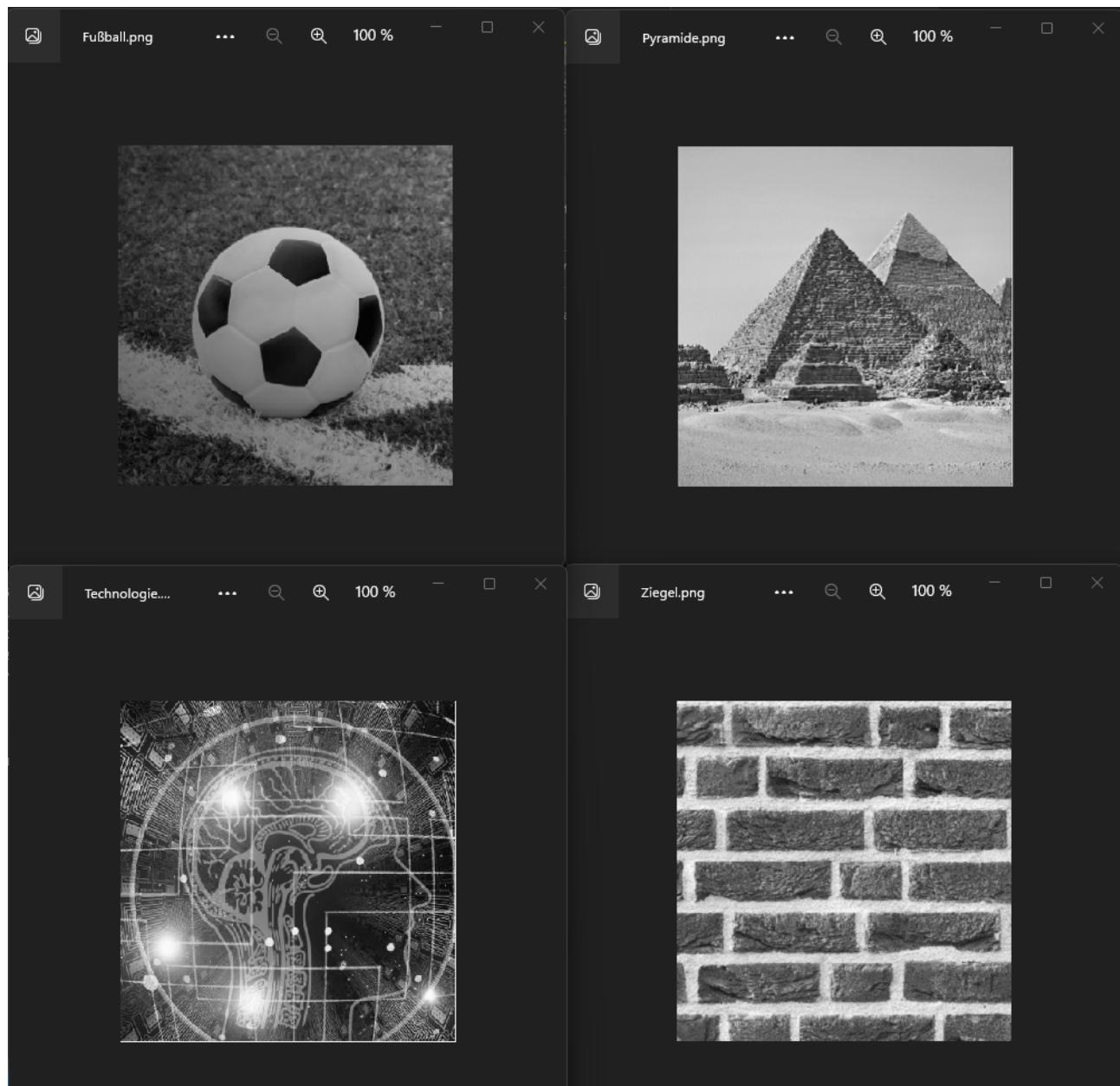


Abbildung 7: Auswahl der Vergleichsbilder



Abbildung 8: Auswahl der Beispielbilder

Zu Beginn wurde noch ein Vergleichsbild genommen, dass eine um einiges geringere Datengröße hatte als die jetzige Auswahl, dadurch wurde immer dieses Bild laut Klassifizierung erkannt. Mit der aktuellen Auswahl von Vergleichsbildern liegt die Datengröße zwischen 65 und 85 KiloByte, was zwar keine optimale Voraussetzung ist, aber zum Klassifizieren der Bilder bereits gute Ergebnisse geliefert hat. Um die Bilder zu klassifizieren, wurde ähnlich vorgegangen wie bei den Texten, indem die Bilder einfach als Text zusammenkopiert und wieder in ein png umgewandelt wurden.

		Ergebnis				
		Testbilder	Technologie	Fußball	Pyramide	Ziegel
T e s t	Technologie	3	4	2	1	
	Fußball	4	2	1	3	
	Pyramide	3	4	1	2	
	Ziegel	3	4	1	2	

Abbildung 9: Konfusionsmatrix der Bilder

Aus der Konfusionsmatrix kann abgeleitet werden, dass die Klassifizierung nicht perfekt ist. Dennoch ergibt sich ein Muster, das die Originalbilder nie an letzter Stelle sind. Vor allem beim Fußball geht dies klar hervor, da dieser sonst immer an letzter Stelle war. Das Technologiebild im Gegensatz hat immer relativ schlecht abgeschnitten, was vermutlich daran liegt, weil das Beispielbild eher schlecht gewählt war. Die "futuristischen" Linien reichen dabei nicht aus, um das Bild zu klassifizieren. Die Pyramide wurde immer gut erkannt, da diese Ähnlichkeiten zu den anderen Bildern aufweist (Linien sowie Weißanteil).

Aus der Konfusionsmatrix kann zwar meiner Meinung nach eine gewisse Klassifizierungsgenauigkeit festgestellt werden, jedoch reicht dies noch nicht aus, um statistisch relevante Daten zu liefern. Dafür müssten noch viel mehr Daten verwendet werden, die noch besser aufgearbeitet worden sind. Trotzdem ist positiv zu erwähnen, dass die eigentlichen Klassen nie auf dem letzten Platz landen und nur einmal auf dem vorletzten Platz.

3 2.3 Kompression und Code-Transformation

3.1 a

Die Codierung mit dem Lempel-Ziv-Verfahren kommt mit der 25-stelligen Sequenz auf eine Kompressionsrate von 1,56. Prinzipiell kann gesagt werden, dass eine Kompressionsrate die größer als 1 ist, gut ist. Die Kompressionsrate ist in diesem Fall schon fast sehr gut, weil die Länge der Nachricht fast halbiert wird.

Sequenz: dabbabababbbbaaaabababccdd

Actual	Next	Wörterbuch?	Wörterbuch	Output
d	a	da -> N	da -> <257>	<100>
a	b	ab -> N	ab -> <258>	<97>
b	b	bb -> N	bb -> <259>	<98>
b	a	ba -> N	ba -> <260>	<98>
a	b	ab -> Y		
ab	a	aba -> N	aba -> <261>	<258>
a	b	ab -> Y		
ab	a	aba -> Y		
aba	b	abab -> N	abab -> <262>	<261>
b	b	bb -> Y		
bb	b	bbb -> N	bbb -> <263>	<259>
b	a	ba -> Y		
ba	a	baa -> N	baa -> <264>	<260>
a	a	aa -> N	aa -> <265>	<97>
a	a	aa -> Y		
aa	b	aab -> N	aab -> <266>	<265>
b	a	ba -> Y		
ba	b	bab -> N	bab -> <267>	<260>
b	a	ba -> Y		
ba	b	bab -> Y		
bab	c	babc -> N	babc -> <268>	<267>
c	c	cc -> N	cc -> <269>	<99>
c	d	cd -> N	cd -> <270>	<99>
d	d	dd -> N	dd -> <271>	<100>
d				<100>

Ursprüngliche Menge der Daten:	25
Neue Menge der Daten	16
Kompressionsrate	1,5625

Abbildung 10: Händische Codierung mit Lempel-Ziv

3.2 b

Bei der Huffman Codierung wurde zuerst die Wahrscheinlichkeit der einzelnen Symbole berechnet. Anschließend wurde der Baum aufgezeichnet und den jeweiligen Zeichen eine Codierung zugeordnet. Zur Huffman Codierung ist generell eine niedrige Entropie der Nachricht von Bedeutung. Eine hohe Homogenität ist wünschenswert aber nicht notwendig. Es ergibt sich eine mittlere Codewortlänge von 1,8 statt 2 Bit und eine Kompressionsrate von 1,11 (größer 1 = gut).

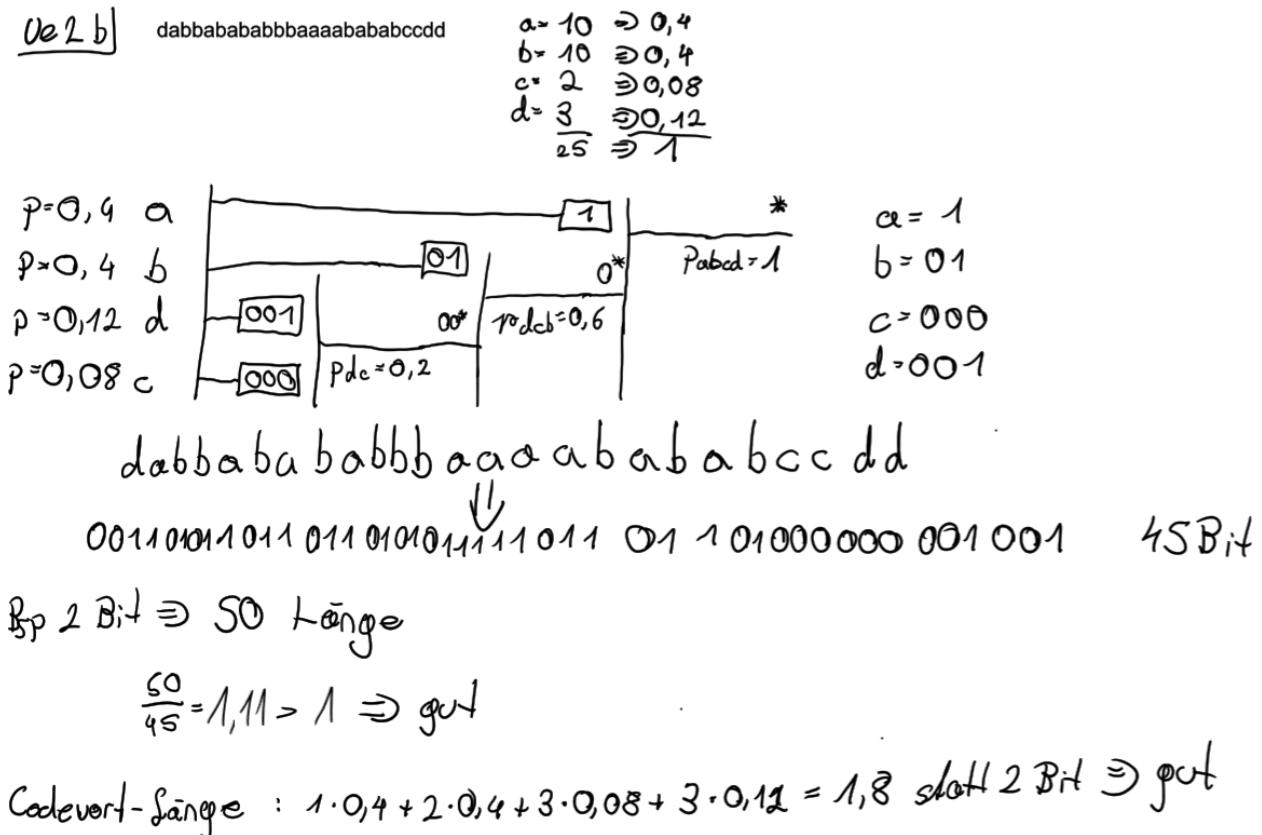


Abbildung 11: Händische Huffman-Codierung

Bei einer Sequenz mit gleichen Zeichen ist die Kompressionsrate maximal. In unserem Beispiel haben wir die Sequenz aaaaaaaaa gewählt. Die Zeichen werden mit 4 Bit codiert, somit erhalten wir eine Gesamtgröße von 40 Bit. Die Wahrscheinlichkeit, mit der a auftritt ist 1, a entspricht im Huffman Coding also 1 und wird mit einem Bit codiert, woraus sich eine Größe der Sequenz von 10 Bit ergibt. Die Kompressionsrate entspricht 40 Bit / 10 Bit, also 4.

Bei einer Sequenz mit komplett unterschiedlichen Zeichen ist die Kompressionsrate minimal. In unserem Beispiel haben wir die Sequenz abcdefghij gewählt. Die Zeichen werden mit 4 Bit codiert, somit erhalten wir wieder eine Gesamtgröße von 40 Bit. Die Wahrscheinlichkeit, mit der jeder Buchstabe auftritt liegt bei 0,1. Im Huffman Tree werden a, b, c und d mit 4 Bit codiert und die anderen Buchstaben mit jeweils 3 Bit. Daraus ergibt sich eine Größe von $4 * 4 + 6 * 3 = 34$. Die Kompressionsrate entspricht hier 40 Bit / 36 Bit, also ca. 1,18.

Bei einer niedrigen Entropie ist die Huffman Codierung am effizientesten. Je vielfältiger die Zeichen sind, desto niedriger ist die Kompressionsrate. Zudem hängt die Kompressionsrate von der mittleren Codewortlänge ab.

3.3 c

Die Runlength Codierung kann erweitert werden, indem statt der reinen Anzahl der Werte, weiters spezifiziert wird um welches Symbol es sich handelt. Dies ist notwendig, da wir nicht mehr nur zwei Zustände haben, sondern eventuell auch mehr. Zusätzlich ist es bei einer größeren Anzahl notwendig Trennzeichen einzusetzen, da ansonsten eventuell nicht mehr differenziert werden kann, ob es sich um ein Symbol oder eine Anzahl handelt. Außerdem muss der Datentyp der codierten Symbole bekannt sein. Wichtig beim Runlength Coding ist eine hohe Homogenität und darauf aufbauend eine niedrige Entropie der Nachricht, weil ansonsten die Nachricht eher vergrößert wird (vor allem bei $n > 2$).

Sequenz	11110101011110000011110001010	Länge:	30
Codiert	41111155431111	Länge:	14
		Kompressionsrate:	2,14
Sequenz	111111222222333333444444555555666666777777888888		48
Codiert	1 6 2 6 3 6 4 6 5 6 6 6 7 6 8 6		31
		Kompressionsrate:	1,55

Abbildung 12: Runlength Codierung

3.4 d

Sequenz: 221111226611122333345645112111

Zahl	1	2	3	4	5	6	ges
Anzahl	12	7	4	2	2	3	30
Wahrscheinlichkeit	0,4	0,23	0,13	0,07	0,07	0,1	1
Entropie	0,53	0,49	0,39	0,26	0,26	0,33	2,26

Abbildung 13: Berechnung der Entropie mittels Logarithmus

Die Entropie wird minimal wenn in der Sequenz nur gleiche Symbole sind z.B. 1111111111. Dabei liegt die Entropie bei 0. Maximal wird die Entropie bei durchgängig unterschiedlichen Zeichen z.B. 0123456789. Dabei liegt die Entropie dann bei ca. 3,3. Die Wahrscheinlichkeit für alle Zeichen ist dann gleich hoch. Eine hohe Entropie, also eine hohe Unordnung bei der Nachricht hat prinzipiell eine eher schlechte Auswirkung auf die Kompressionsrate. Allerdings wirkt auch die Homogenität einer Nachricht auf die Kompressionsrate ein. Homogenität beschreibt die Ähnlichkeit der Nachbarn, also die lokale Korrelation der Symbole.