VYTAUTAS MAGNUS UNIVERSITY
FACULTY OF INFORMATICS

Paulius Lapienis

DIGITAL IMAGE PROCESSING
LAB WORK NO. 3

Kaunas, 2023

CONTENT

ABSTRACT

This work shows how changing the parameters for Eigenfaces, Fisherfaces and Local Binary Pattern affects face recognition. The experiments performed follows this order:

1. Images are loaded using OpenCV.
2. Images are converted to grayscale.
3. Each Image is loaded into the Haar cascades face recognizer.
4. The recognizer is used to mark spots where a face is detected and extract the faces.
5. The recognized faces are used to train face recognition models.
6. The images then receive noise.
7. The model performance using different parameters is then checked.
8. Data is aggregated to tables.
9. The data is visualized in a graph.
10. Results/Conclusions are made.

1 TASK: PROCESSING

1.1 Image preparation

Image used for transformation was obtained from the internet. The chosen image contained 4 faces.

Images were then loaded using the OpenCV library [1] and transformed to gray-scale using the cvtColor function.

```
cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```
(1)

These images were then used to detect images using the Haar Cascade classifier. To set the scale_factor an min_neighbours parameters the best values from lab. work no. 2 were chosen (1.3 and 5.0 respectively).

```
faces = face_cascade.detectMultiScale(input_image, scale_factor, min_neighbours)
```
(2)

The face detection using Haar Cascade classifier can be seen in Figure 1. Figure 2. Contains an example of the noise (speckle) added when testing the model performance. To add the noise the following code was used:

```
row, col = image.shape
gauss = np.random.randn(row, col)
gauss = gauss.reshape(row, col)
return image + image * gauss
```
(3)

a) b) c)

Figure 1. Image used in experiments: a) original, b) grayscale, c) with faces detected

a) b) c)

Figure 2. Image used in experiments: a) original, b) grayscale, c) noisy

1.2 Parameter space exploration

To explore the parameter space and find the optimal value a Bayesian optimisation strategy was used [2]. A strategy with 5 initial random points and 100 subsequent exploration points was chosen. Black box functions that contained the initialized model with parameters from the strategy was created. An example of such a function:

```
def eigen_bbox(num_components: float, threshold: float):                    (4)
    num_components, threshold = int(num_components), int(threshold)
    model = cv2.face.EigenFaceRecognizer_create(
        num_components=num_components, threshold=threshold
    )


    return perform_experiment(
        model,
        DATA_OUTPUT_DIR / "eigen",
        num_components=num_components,
        threshold=threshold,
    )


return eigen_bbox
```

Here the perform experiment function trains the model, and uses it on the image with noise to detect faces. It also saves the results of each experiment in both images and tables which are later transformed to a final table and used for visualization purposes. The function finally returns a sum of the confidences for correctly guessed labels which is then used for the Bayesian optimization.

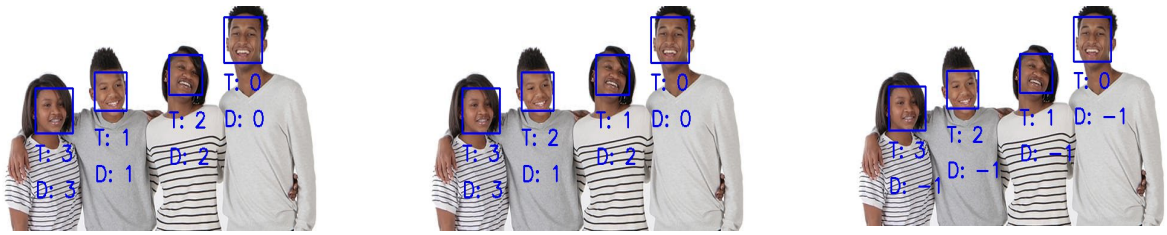
```
@Counter                                                                    (5)
def perform_experiment(model, data_dir: Path, **kwargs: int):
    model.train(np.asarray(xl), np.asarray(yl))
    img = cv2.imread(str((DATA_INPUT_DIR / "input_1.jpg")))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = noisy("speckle", gray)
    face_cascade = cv2.CascadeClassifier(
        str(HAAR_CASCADES_DIR / "haarcascade_frontalface_default.xml")
    )
    faces = face_cascade.detectMultiScale(img, 1.3, 3)
    rows = []
    for i, (x, y, w, h) in enumerate(faces):
        img = cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
        f = cv2.resize(gray[y : y + h, x : x + w], (200, 200))
        try:
            params = model.predict(f)
            rows.append(
                {
                    "label": params[0],
```

```
                "true_label": i,
                "confidence": params[1],
            }
            | kwargs
        )
        cv2.putText(
            img, f"T: {i}", (x, y + 100), cv2.FONT_HERSHEY_SIMPLEX, 1, 255, 2
        )
        cv2.putText(
            img,
            f"D: {params[0]}",
            (x, y + 150),
            cv2.FONT_HERSHEY_SIMPLEX,
            1,
            255,
            2,
        )
    except Exception:
        traceback.print_exc()
cv2.imwrite(
    f"{data_dir}/experiment_{'_'.join(str(val) for val in kwargs.values())}.png",
    img,
)
df = pd.DataFrame(rows)
df.to_csv(
    f"{data_dir}/experiment_{'_'.join(str(val) for val in kwargs.values())}.csv",
    index=False,
)
return df[df["label"] == df["true_label"]]["confidence"].sum()
```

2 EXPERIMENTS

The results of the experiments can be seen in Tables 1, 2, 3. Each table represents experiments performed one method (Eigenfaces, Fisherfaces or Local Binary Pattern). The tables are then further visualized in each subsection.

Bellow is an example of a good, bad and medium result visualization.



a) b) c)

Figure 3. Results: a) good, b) medium, c) bad. T – true label, D – detected label.

Each parameter was bound to a space. The boundaries can be seen in the code bellow:

```
threshold_bounds = (0, 100000)                                    (6)
other_bounds = (1, 1000)


pbounds = {
   Method.EIGEN: {"num_components": other_bounds, "threshold": threshold_bounds},
   Method.FISHER: {"num_components": other_bounds, "threshold": threshold_bounds},
   Method.LBPH: {
      "radius": (1, 100),
      "grid": other_bounds,
      "threshold": threshold_bounds,
   },
}
```

2.1 Eigenfaces

Table 1. Results from the experiments. # means the value was inf. Rows marked with green are rows where all of the values were correct, while rows marked with red mean all the values were incorrect.

experiment	correct	incorrect	unrecognized	total	max_confidence	sum_confidence_correct	num_components	threshold
0	2	2	0	4	967	855	491	96727
1	2	2	0	4	1174	2112	166	60278
2	4	0	0	4	937	2260	939	74024

3	2	2	0	4	1306	2122	192	4176
4	4	0	0	4	1028	3399	100	75057
5	1	3	0	4	1521	1521	77	75760
6	1	3	0	4	903	757	954	74015
7	2	2	0	4	1581	1749	671	99505
8	2	2	0	4	1647	1391	483	40216
9	2	2	0	4	1267	1560	653	87892
10	2	2	0	4	1448	2756	927	74020
11	2	2	0	4	788	1292	559	40087
12	2	2	0	4	1563	2282	550	76975
13	2	2	0	4	1901	1735	122	75070
14	2	2	0	4	1619	1125	559	76984
15	2	2	0	4	1076	1066	177	4184
16	2	2	0	4	1319	1565	844	32015
17	1	3	0	4	1338	450	342	36980
18	2	2	0	4	1280	2143	108	75054
19	2	2	0	4	1223	1300	116	75049
20	1	3	0	4	1143	1143	901	44581
21	2	2	0	4	1541	1431	544	76954
22	2	2	0	4	2595	1766	932	74010

23	2	2	0	4	1790	1117	176	60258
24	2	2	0	4	672	1230	717	5389
25	2	2	0	4	1452	2800	148	75071
26	2	2	0	4	1772	1614	928	82817
27	2	2	0	4	1595	1229	736	61558
28	2	2	0	4	985	1905	102	75086
29	2	2	0	4	992	1129	408	88431
30	2	2	0	4	1946	827	107	75083
31	2	2	0	4	1099	1189	447	5063
32	2	2	0	4	1508	1827	582	41734
33	4	0	0	4	1472	3674	504	14466
34	2	2	0	4	1252	1374	114	6836
35	4	0	0	4	745	2534	742	21906
36	2	2	0	4	1521	1420	190	10149
37	1	3	0	4	905	799	200	2918
38	2	2	0	4	1163	2046	645	25789
39	2	2	0	4	1549	1247	707	40885
40	2	2	0	4	955	1149	572	80363
41	2	2	0	4	1567	2286	661	35527
42	2	2	0	4	1115	729	194	97723

43	2	2	0	4	1705	2010	920	70355
44	0	0	4	4	#	0	456	274
45	2	2	0	4	1189	1654	516	76958
46	2	2	0	4	1104	586	597	89340
47	2	2	0	4	1254	1524	69	37394
48	2	2	0	4	930	1853	218	50412
49	2	2	0	4	1445	2015	737	69432
50	0	0	4	4	#	0	77	155
51	2	2	0	4	1565	2368	26	22495
52	2	2	0	4	1095	1127	654	14017
53	2	2	0	4	1364	932	727	69429
54	2	2	0	4	1318	1699	350	25011
55	2	2	0	4	1603	1846	938	62719
56	2	2	0	4	1241	1665	968	53527
57	2	2	0	4	1163	1458	456	82738
58	2	2	0	4	1605	1144	290	4242
59	1	0	3	4	#	454	503	532
60	2	2	0	4	1748	2169	362	35827
61	2	2	0	4	1370	2004	182	85322
62	2	2	0	4	1022	1391	490	3020

63	2	2	0	4	1049	803	137	80893
64	2	2	0	4	1062	2028	796	12966
65	2	2	0	4	1074	1509	166	90986
66	2	2	0	4	955	1430	752	90011
67	2	2	0	4	1381	764	134	18100
68	2	2	0	4	982	1097	703	79631
69	2	2	0	4	1019	1250	45	38005
70	2	2	0	4	1235	1453	621	97761
71	1	3	0	4	2164	737	654	61427
72	2	2	0	4	1395	1784	429	73228
73	2	2	0	4	1187	2284	974	2277
74	1	3	0	4	1505	782	175	63559
75	2	2	0	4	890	1565	956	72379
76	1	3	0	4	819	500	566	97374
77	2	2	0	4	1712	1873	399	39412
78	2	2	0	4	1101	1203	378	1760
79	2	2	0	4	948	785	738	69428
80	2	2	0	4	827	1468	7	25994
81	2	2	0	4	1685	929	618	74853
82	2	2	0	4	1533	2961	381	80490

83	2	2	0	4	1061	650	6	38954
84	2	2	0	4	1279	1573	311	62451
85	2	2	0	4	1440	2712	392	59984
86	2	2	0	4	1118	1424	519	92412
87	2	2	0	4	1449	2075	323	47312
88	2	2	0	4	1116	1693	396	25784
89	2	2	0	4	831	1451	937	48479
90	4	0	0	4	1292	3306	961	57631
91	2	2	0	4	1019	1802	732	84844
92	2	2	0	4	1378	2244	833	3780
93	2	2	0	4	1707	2198	274	54313
94	1	2	1	4	#	666	437	803
95	2	2	0	4	1760	1510	601	58588
96	2	2	0	4	1384	1032	584	51589
97	2	2	0	4	1898	2007	688	33520
98	2	2	0	4	1924	1103	956	28072
99	2	2	0	4	1004	1769	732	23901
100	1	3	0	4	1014	1014	272	78221
101	2	2	0	4	951	1804	127	65232
102	2	2	0	4	998	1874	315	67050

103	2	2	0	4	1316	1415	943	86900
104	2	2	0	4	1574	1075	819	19169

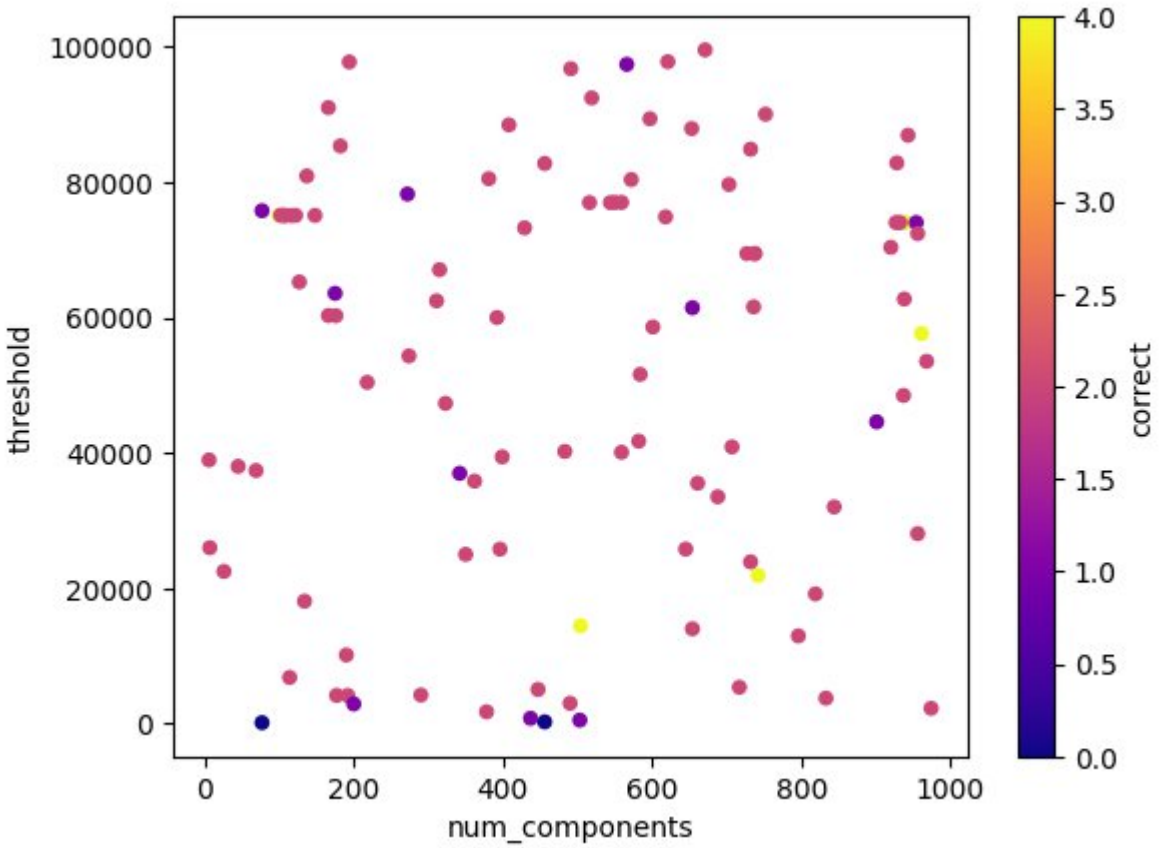


Figure 4. Scatterplot of the correct image count based on threshold and num_components.

2.2 Fisherfaces

Table 2. Results from the experiments. # means the value was inf. Rows marked with green are rows where all of the values were correct, while rows marked with red mean all the values were incorrect.

experiment	**correct**	**incorrect**	**unrecognized**	**total**	**max_confidence**	**sum_confidence_correct**	**num_components**	**threshold**
0	4	0	0	4	983	3018	535	31084
1	4	0	0	4	1628	2542	700	75710
2	2	2	0	4	1119	1614	292	35294

3	1	3	0	4	1318	1318	119	52031
4	2	2	0	4	1459	1854	786	47919
5	2	2	0	4	1152	2020	761	62534
6	1	1	2	4	#	129	423	366
7	1	3	0	4	948	262	190	43154
8	1	3	0	4	1212	597	887	75964
9	4	0	0	4	635	1803	637	67829
10	2	2	0	4	1562	1221	809	83245
11	2	2	0	4	1064	1526	226	73205
12	2	2	0	4	580	992	911	60591
13	2	2	0	4	1470	1425	263	99633
14	2	2	0	4	1229	1637	592	94646
15	2	2	0	4	1529	1030	374	99383
16	4	0	0	4	1635	3753	188	96290
17	2	2	0	4	1174	1949	217	96320
18	4	0	0	4	1182	3270	891	36466
19	2	2	0	4	1080	1309	178	61630
20	2	2	0	4	732	1322	898	4000
21	2	2	0	4	1166	1572	311	14187
22	2	2	0	4	2016	2276	309	88413

23	2	2	0	4	1681	1265	880	36465
24	1	3	0	4	933	933	677	99233
25	2	2	0	4	1685	1056	322	71988
26	2	2	0	4	1515	2068	905	36278
27	2	2	0	4	1508	1221	417	98630
28	1	3	0	4	2233	509	883	65295
29	4	0	0	4	1122	3628	1000	35075
30	2	2	0	4	1122	1195	390	65256
31	2	2	0	4	742	665	449	37677
32	2	2	0	4	925	1778	212	85588
33	4	0	0	4	1988	3427	995	16609
34	2	2	0	4	430	530	621	97176
35	2	2	0	4	963	1053	420	78426
36	2	2	0	4	1163	2124	266	76893
37	2	2	0	4	670	1323	201	85585
38	2	2	0	4	1095	1670	260	76862
39	1	3	0	4	1113	553	682	90644
40	2	2	0	4	539	589	653	31282
41	2	2	0	4	1401	1264	639	74453
42	2	2	0	4	863	1212	800	23548

43	2	2	0	4	2664	1302	505	58934
44	2	2	0	4	624	911	882	1029
45	2	2	0	4	1008	1159	562	45384
46	2	2	0	4	1821	2820	201	85577
47	2	2	0	4	1609	1075	195	30445
48	2	2	0	4	773	1341	67	38654
49	1	3	0	4	933	253	737	40076
50	2	2	0	4	1985	2443	206	85606
51	1	3	0	4	1723	1723	204	3419
52	2	2	0	4	886	1763	371	39701
53	2	2	0	4	909	1318	163	80742
54	2	2	0	4	942	1017	493	69375
55	2	2	0	4	519	1026	673	41656
56	2	2	0	4	599	899	897	31141
57	2	2	0	4	1034	1651	703	77228
58	2	2	0	4	862	1498	347	55557
59	2	2	0	4	1004	879	770	85316
60	2	2	0	4	689	1265	137	89314
61	1	3	0	4	1319	788	104	37459
62	2	2	0	4	907	1474	786	11448

63	2	2	0	4	1139	807	279	52325
64	2	2	0	4	1100	1785	632	93933
65	2	2	0	4	1487	1908	467	69050
66	2	2	0	4	970	993	916	54626
67	2	2	0	4	1810	1092	285	96976
68	1	3	0	4	998	765	537	59563
69	2	2	0	4	1040	1130	47	27211
70	2	2	0	4	1223	1950	266	76886
71	2	2	0	4	1031	1145	660	24746
72	2	2	0	4	971	1510	289	76883
73	2	2	0	4	582	1100	631	93915
74	2	2	0	4	1444	2027	250	76875
75	2	2	0	4	1136	1500	294	76896
76	2	2	0	4	1473	807	382	39712
77	1	3	0	4	346	256	188	3436
78	2	2	0	4	1057	629	372	39703
79	2	2	0	4	1080	1548	878	5521
80	2	2	0	4	1049	1753	371	77017
81	1	3	0	4	831	124	665	71361
82	2	2	0	4	957	1443	331	68370

83	2	2	0	4	887	1207	165	41443
84	2	2	0	4	1186	406	865	40708
85	2	2	0	4	1241	1040	902	50097
86	2	2	0	4	1677	1846	626	4934
87	2	2	0	4	753	1192	510	42460
88	1	3	0	4	1471	904	315	55780
89	2	2	0	4	661	528	372	13560
90	2	2	0	4	1190	1492	470	92750
91	2	2	0	4	674	628	938	4061
92	2	2	0	4	1447	2799	768	59636
93	2	2	0	4	833	1389	454	24086
94	2	2	0	4	1037	1258	834	48391
95	2	2	0	4	984	1342	858	83289
96	2	2	0	4	932	1744	662	85445
97	2	2	0	4	991	1559	10	15341
98	2	2	0	4	1496	1633	773	91263
99	2	2	0	4	1135	1425	499	83424
100	2	2	0	4	927	1805	644	44203
101	2	2	0	4	948	1813	117	78893
102	2	2	0	4	1309	1500	819	11792

103	2	2	0	4	980	1205	54	55625
104	2	2	0	4	1118	1269	196	62483
105	2	2	0	4	783	422	794	2588
106	2	2	0	4	1047	1596	530	4644
107	2	2	0	4	808	975	878	82850
108	1	3	0	4	1543	410	515	92286
109	2	2	0	4	1771	750	874	78677
110	2	2	0	4	1493	1897	92	35702
111	2	2	0	4	687	869	686	53481
112	2	2	0	4	1563	2669	113	17269
113	2	2	0	4	1154	1167	754	31327
114	2	2	0	4	581	943	460	68781
115	1	3	0	4	869	782	377	90137
116	2	2	0	4	1413	1576	515	28767
117	2	2	0	4	720	1286	960	41528
118	2	2	0	4	1213	1955	696	9161
119	2	2	0	4	1347	1034	192	59575
120	2	2	0	4	936	1510	126	97837
121	2	2	0	4	1163	1652	751	29918
122	2	2	0	4	1582	1172	505	78512

123	2	2	0	4	1215	2202	772	31469
124	2	2	0	4	1194	1708	564	56167
125	2	2	0	4	1578	2503	236	53348
126	1	2	1	4	#	600	935	690
127	2	2	0	4	876	1484	222	41210
128	1	3	0	4	1133	693	521	81289
129	2	2	0	4	1158	2017	487	80740
130	2	2	0	4	761	1075	105	85608
131	2	2	0	4	893	909	646	25077
132	2	2	0	4	1573	2002	978	2972
133	2	2	0	4	1349	1672	20	18250
134	2	2	0	4	1131	852	111	1602

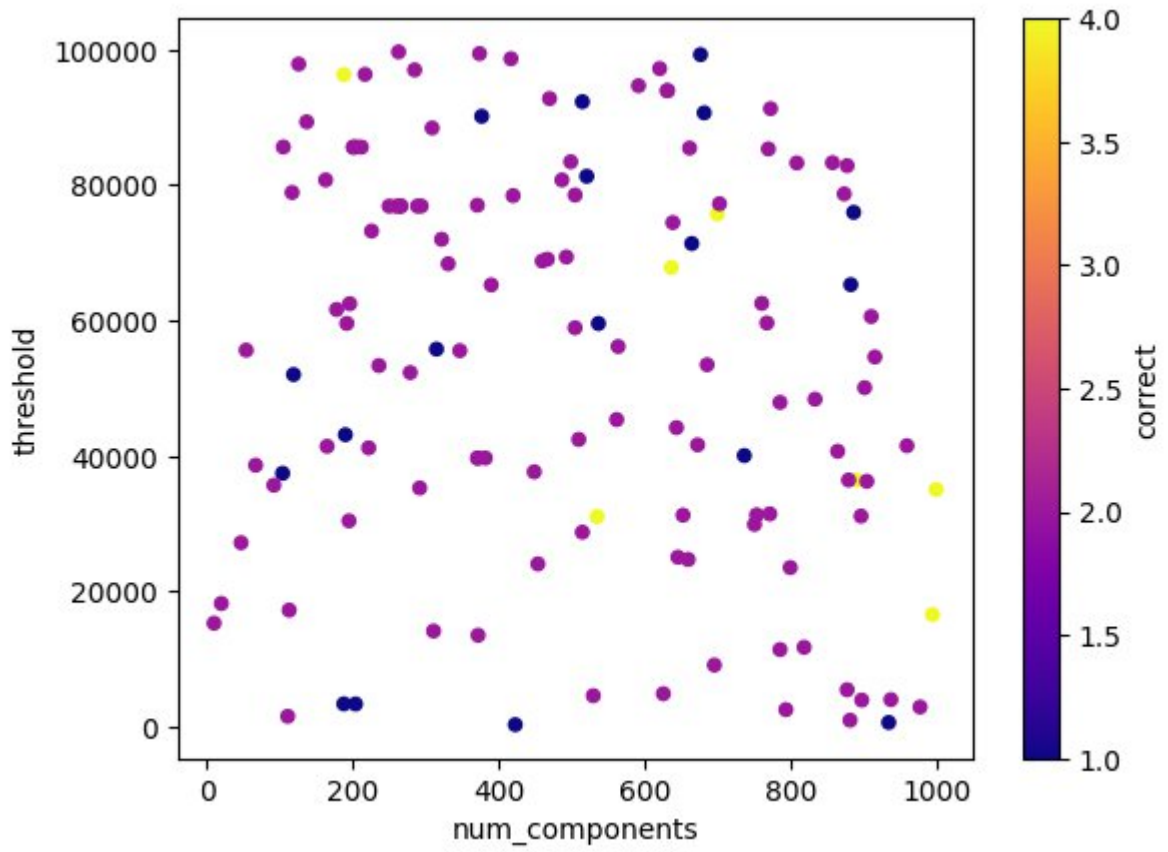


Figure 5. Scatterplot of the correct image count based on threshold and num_components.

2.3 Local Binarry Pattern

Table 3. Results from the experiments. # means the value was inf. Rows marked with green are rows where all of the values were correct, while rows marked with red mean all the values were incorrect.

experiment	correct	incorrect	unrecognized	total	max_confidence	sum_confidence_correct	radius	grid	threshold
0	1	3	0	4	0	0	10	550	69529
1	0	0	4	4	#	0	3	169	45087
2	1	3	0	4	0	0	64	572	38088
3	1	3	0	4	0	0	95	410	75637
4	1	3	0	4	0	0	64	562	86825
5	1	3	0	4	0	0	54	124	7
6	1	3	0	4	0	0	96	74	99905

7	1	3	0	4	2	2	76	1	15423
8	1	3	0	4	0	0	71	243	91435
9	1	3	0	4	0	0	81	140	70210
10	1	3	0	4	0	0	81	987	20593
11	2	2	0	4	5612	10724	80	38	15466
12	1	3	0	4	0	0	76	972	69635
13	1	3	0	4	0	0	33	345	46739
14	1	3	0	4	0	0	79	179	28877
15	1	3	0	4	0	0	80	751	82279
16	1	3	0	4	0	0	32	850	54739
17	1	3	0	4	0	0	47	108	18304
18	1	3	0	4	0	0	45	725	89601
19	1	3	0	4	0	0	71	99	63164
20	1	3	0	4	0	0	31	618	71066
21	1	3	0	4	0	0	86	864	56209
22	1	3	0	4	0	0	77	557	70666
23	1	3	0	4	0	0	33	887	61447
24	1	3	0	4	8098	7929	10	46	42911
25	1	3	0	4	0	0	46	769	20927
26	1	3	0	4	0	0	92	174	97113

27	1	3	0	4	0	0	67	90	71791
28	1	3	0	4	0	0	19	437	6303
29	1	3	0	4	0	0	6	550	3581
30	1	3	0	4	0	0	43	384	63443
31	1	3	0	4	0	0	76	297	84844
32	1	3	0	4	0	0	48	428	82712
33	2	2	0	4	7391	14664	23	45	42897
34	1	3	0	4	165	158	7	8	15444
35	1	3	0	4	0	0	22	856	10869
36	2	2	0	4	21872	42692	32	76	42912
37	1	3	0	4	0	0	95	227	79622
38	1	3	0	4	0	0	70	203	87717
39	2	2	0	4	4030	7838	21	33	42875
40	1	3	0	4	0	0	63	853	78979
41	1	3	0	4	0	0	90	806	93658
42	2	2	0	4	1861	3644	77	22	42953
43	1	3	0	4	0	0	93	68	42902
44	1	3	0	4	0	0	81	564	87884
45	2	2	0	4	2240	4390	44	25	42941
46	2	2	0	4	21224	41376	58	74	42904

47	1	3	0	4	0	0	87	157	49657
48	1	3	0	4	0	0	95	160	46792
49	1	3	0	4	0	0	27	501	12510
50	1	3	0	4	0	0	70	128	15381
51	1	3	0	4	26273	25742	14	83	42942
52	1	3	0	4	0	0	83	316	24316
53	1	3	0	4	0	0	45	113	42984
54	1	3	0	4	0	0	74	716	53859
55	0	0	4	4	#	0	8	133	42874
56	1	3	0	4	0	0	65	623	9298
57	1	3	0	4	0	0	39	153	42925
58	3	1	0	4	16380	48640	63	65	42846
59	1	3	0	4	0	0	71	687	59395
60	1	3	0	4	21204	21204	42	75	42965
61	1	3	0	4	0	0	65	95	42944
62	1	3	0	4	0	0	47	885	95053
63	1	3	0	4	0	0	33	799	31301
64	1	3	0	4	0	0	75	427	53269
65	1	3	0	4	3776	3644	74	31	15490
66	1	3	0	4	0	0	77	689	12359

67	2	2	0	4	11057	21777	9	54	42912
68	1	3	0	4	6888	6772	63	42	42854
69	2	2	0	4	30672	60980	18	89	42877
70	1	3	0	4	0	0	54	878	74704
71	4	0	0	4	28412	112464	52	86	42967
72	1	3	0	4	0	0	94	56	42966
73	2	2	0	4	10864	21616	64	53	42839
74	1	3	0	4	0	0	11	423	85497
75	1	3	0	4	22596	22268	33	77	42908
76	2	2	0	4	7452	14241	30	45	42822
77	1	3	0	4	0	0	55	96	42818
78	1	3	0	4	0	0	59	98	42865
79	1	3	0	4	0	0	73	61	15474
80	1	3	0	4	0	0	75	99	42971
81	1	3	0	4	0	0	68	95	42985
82	1	3	0	4	0	0	86	104	52983
83	1	3	0	4	0	0	98	662	58526
84	2	2	0	4	14159	27791	20	61	42752
85	2	2	0	4	4136	8129	14	34	42718
86	2	2	0	4	16508	32524	40	66	42721

87	1	3	0	4	0	0	87	39	15438
88	2	2	0	4	21624	42760	55	75	42743
89	1	3	0	4	10000	9908	50	51	42709
90	2	2	0	4	10118	19863	48	52	42907
91	2	2	0	4	28468	55516	30	86	42971
92	2	2	0	4	13068	25900	68	58	42905
93	2	2	0	4	1784	3500	50	22	42834
94	2	2	0	4	13685	26568	25	60	42869
95	2	2	0	4	40228	79680	43	103	42744
96	2	2	0	4	16844	33208	39	67	42697
97	2	2	0	4	29204	58056	50	87	42907
98	1	3	0	4	0	0	81	70	42765
99	2	2	0	4	24276	48300	45	80	42687
100	1	3	0	4	0	0	55	104	42736
101	2	2	0	4	35604	70020	37	97	42974
102	2	2	0	4	23380	45352	43	78	42937
103	1	3	0	4	39828	39828	32	102	42874
104	2	2	0	4	25936	51224	32	83	42679

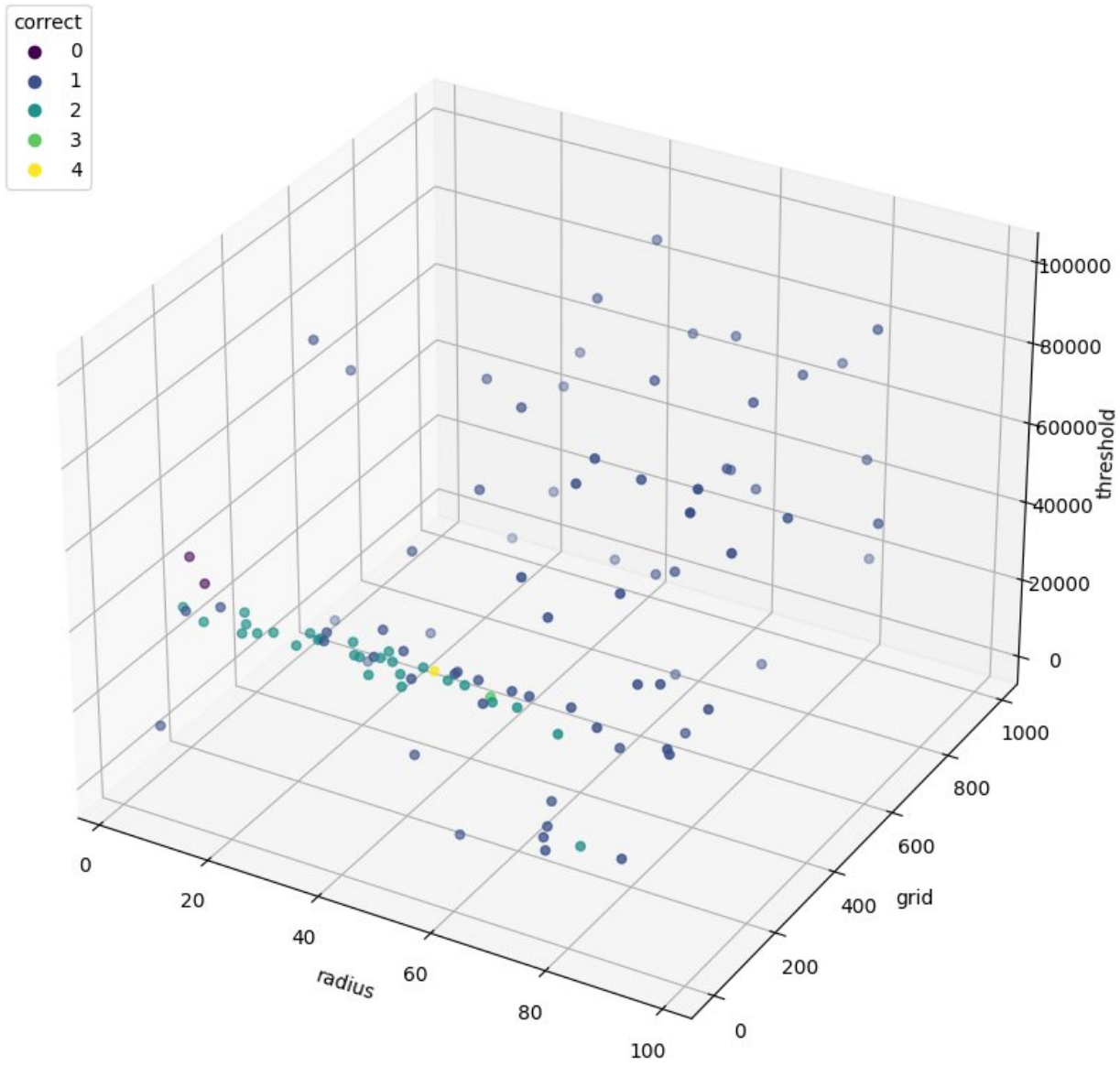


Figure 6. Scatterplot of the correct image count based on threshold and radius and grid.

3 RESULTS AND CONSLUSIONS

3.1 Results

1. Eigenfaces method had on average 48.8% accuracy. It achieved 100% accuracy with (939, 74024) (100, 75057) (504, 14466) (742, 21906) (961, 57631) where the first number is num_components and the second number is the threshold (5 instances). Similarly 0% accuracy was achieved with (456, 274) (77, 155) (2 instances).
2. Fisherfaces method had on average 49.3% accuracy. It achieved 100% accuracy with (535, 31084) (700, 75710) (637, 67829) (188, 96290) (891, 36466) (1000 35075) (995, 16609) where the first number is num_components and the second number is the threshold (7 instances). 0 % accuracy was never achieved in any of the experiments (0 instances).
3. Local Binary pattern method had on average 32.1% accuracy. It achieved 100% accuracy with (52, 86, 42967) where the first number is the radius, second is the grid and third is the threshold parameter (1 instance). 0 % accuracy was achieved with (3, 169, 45087) (8, 133, 42874) (2 instances).
4. Overall around 100 experiments were performed for each method.

3.2 Conclusions

1. The best method is Fisherfaces. It has the highest average accuracy (49.3%), most instances of 100% accuracy (7) and least instances with 0% accuracy (0).
2. The worst method is Local Binary pattern. It has the lowest average accuracy (32.1%). It has lowest instances of 100% accuracy achieved (1) and highest amount of instances with 0% accuracy achieved (2).

4 REFERENCES

[1] opencv-python (n.d.) OpenCV Team. Retrieved from https://pypi.org/project/opencv-python/

[2] Bayesian Optimization: Open source constrained global optimization tool for Python. Retrieved from https://github.com/fmfn/BayesianOptimization