A study investigates different methods to boost performance of neural networks when performing sentiment analysis on IMDb dataset data. Model optimization requires different approaches that modify architecture designs alongside function improvement and regularization technique application.

Architectural Adjustments The study examines the impact that changing hidden layer numbers has on modeling outcomes. The model performance benefits from both the number of hidden layers and the units present within these layers.

Functional Modifications The experimental studies to determine how loss functions affect model accuracy levels and running speed during the training process. Selecting alternative activation functions results in better nonlinearity which leads to a more efficient learning process.

Regularization Techniques The regularization strategy incorporates dropout techniques which solve both overfitting problems and generalization tasks.

A total of 50,000 movie reviews from IMDb have been equally distributed between negative and positive expressions. The system conducts training operations on 25,000 reviews before testing with the other 25,000.

The study analyzes systematic modifications of neural networks to discover the best configuration which improves sentiment detection accuracy in movie reviews.

```
from numpy.random import seed
seed(123)
from tensorflow.keras.datasets import imdb
(tr_set, tr_labels), (te_set, te_labels) = imdb.load_data(
    num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
tr_set
```

```
array([list([1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838,
112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447,
4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4,
22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130,
12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4,
107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4,
381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51,
36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178,
32]),
       list([1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14, 394, 20, 13,
119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647,
4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45,
43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605,
2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11,
220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9,
6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285,
16, 145, 95]),
       list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 13, 71, 149, 14, 22, 112, 4, 2401, 311, 12,
16, 3711, 33, 75, 43, 1829, 296, 4, 86, 320, 35, 534, 19, 263, 4821, 1301, 4, 1873, 33, 89, 78, 12, 66, 16, 4, 360, 7,
4, 58, 316, 334, 11, 4, 1716, 43, 645, 662, 8, 257, 85, 1200, 42, 1228, 2578, 83, 68, 3912, 15, 36, 165, 1539, 278, 36,
69, 2, 780, 8, 106, 14, 6905, 1338, 18, 6, 22, 12, 215, 28, 610, 40, 6, 87, 326, 23, 2300, 21, 23, 22, 12, 272, 40, 57,
31, 11, 4, 22, 47, 6, 2307, 51, 9, 170, 23, 595, 116, 595, 1352, 13, 191, 79, 638, 89, 2, 14, 9, 8, 106, 607, 624, 35,
534, 6, 227, 7, 129, 113]),
       ...,
       list([1, 11, 6, 230, 245, 6401, 9, 6, 1225, 446, 2, 45, 2174, 84, 8322, 4007, 21, 4, 912, 84, 2, 325, 725, 134,
2, 1715, 84, 5, 36, 28, 57, 1099, 21, 8, 140, 8, 703, 5, 2, 84, 56, 18, 1644, 14, 9, 31, 7, 4, 9406, 1209, 2295, 2,
1008, 18, 6, 20, 207, 110, 563, 12, 8, 2901, 2, 8, 97, 6, 20, 53, 4767, 74, 4, 460, 364, 1273, 29, 270, 11, 960, 108,
45, 40, 29, 2961, 395, 11, 6, 4065, 500, 7, 2, 89, 364, 70, 29, 140, 4, 64, 4780, 11, 4, 2678, 26, 178, 4, 529, 443, 2,
5, 27, 710, 117, 2, 8123, 165, 47, 84, 37, 131, 818, 14, 595, 10, 10, 61, 1242, 1209, 10, 10, 288, 2260, 1702, 34,
2901, 2, 4, 65, 496, 4, 231, 7, 790, 5, 6, 320, 234, 2766, 234, 1119, 1574, 7, 496, 4, 139, 929, 2901, 2, 7750, 5,
4241, 18, 4, 8497, 2, 250, 11, 1818, 7561, 4, 4217, 5408, 747, 1115, 372, 1890, 1006, 541, 9303, 7, 4, 59, 2, 4, 3586,
2]),
       list([1, 1446, 7079, 69, 72, 3305, 13, 610, 930, 8, 12, 582, 23, 5, 16, 484, 685, 54, 349, 11, 4120, 2959, 45,
58, 1466, 13, 197, 12, 16, 43, 23, 2, 5, 62, 30, 145, 402, 11, 4131, 51, 575, 32, 61, 369, 71, 66, 770, 12, 1054, 75,
100, 2198, 8, 4, 105, 37, 69, 147, 712, 75, 3543, 44, 257, 390, 5, 69, 263, 514, 105, 50, 286, 1814, 23, 4, 123, 13,
161, 40, 5, 421, 4, 116, 16, 897, 13, 2, 40, 319, 5872, 112, 6700, 11, 4803, 121, 25, 70, 3468, 4, 719, 3798, 13, 18,
31, 62, 40, 8, 7200, 4, 2, 7, 14, 123, 5, 942, 25, 8, 721, 12, 145, 5, 202, 12, 160, 580, 202, 12, 6, 52, 58, 2, 92,
401, 728, 12, 39, 14, 251, 8, 15, 251, 5, 2, 12, 38, 84, 80, 124, 12, 9, 23]),
       list([1, 17, 6, 194, 337, 7, 4, 204, 22, 45, 254, 8, 106, 14, 123, 4, 2, 270, 2, 5, 2, 2, 732, 2098, 101, 405,
39, 14, 1034, 4, 1310, 9, 115, 50, 305, 12, 47, 4, 168, 5, 235, 7, 38, 111, 699, 102, 7, 4, 4039, 9245, 9, 24, 6, 78,
1099, 17, 2345, 2, 21, 27, 9685, 6139, 5, 2, 1603, 92, 1183, 4, 1310, 7, 4, 204, 42, 97, 90, 35, 221, 109, 29, 127, 27,
118, 8, 97, 12, 157, 21, 6789, 2, 9, 6, 66, 78, 1099, 4, 631, 1191, 5, 2642, 272, 191, 1070, 6, 7585, 8, 2197, 2, 2,
544, 5, 383, 1271, 848, 1468, 2, 497, 2, 8, 1597, 8778, 2, 21, 60, 27, 239, 9, 43, 8368, 209, 405, 10, 10, 12, 764, 40,
4, 248, 20, 12, 16, 5, 174, 1791, 72, 7, 51, 6, 1739, 22, 4, 204, 131, 9])],
      dtype=object)
```

```
tr_labels[0]
```

```
1
```

```
len(tr_labels)
```

```
25000
```

```
te_set
```

```
array([list([1, 591, 202, 14, 31, 6, 717, 10, 10, 2, 2, 5, 4, 360, 7, 4, 177, 5760, 394, 354, 4, 123, 9, 1035, 1035,
1035, 10, 10, 13, 92, 124, 89, 488, 7944, 100, 28, 1668, 14, 31, 23, 27, 7479, 29, 220, 468, 8, 124, 14, 286, 170, 8,
157, 46, 5, 27, 239, 16, 179, 2, 38, 32, 25, 7944, 451, 202, 14, 6, 717]),
       list([1, 14, 22, 3443, 6, 176, 7, 5063, 88, 12, 2679, 23, 1310, 5, 109, 943, 4, 114, 9, 55, 606, 5, 111, 7, 4,
139, 193, 273, 23, 4, 172, 270, 11, 7216, 2, 4, 8463, 2801, 109, 1603, 21, 4, 22, 3861, 8, 6, 1193, 1330, 10, 10, 4,
105, 987, 35, 841, 2, 19, 861, 1074, 5, 1987, 2, 45, 55, 221, 15, 670, 5304, 526, 14, 1069, 4, 405, 5, 2438, 7, 27, 85,
108, 131, 4, 5045, 5304, 3884, 405, 9, 3523, 133, 5, 50, 13, 104, 51, 66, 166, 14, 22, 157, 9, 4, 530, 239, 34, 8463,
2801, 45, 407, 31, 7, 41, 3778, 105, 21, 59, 299, 12, 38, 950, 5, 4521, 15, 45, 629, 488, 2733, 127, 6, 52, 292, 17, 4,
6936, 185, 132, 1988, 5304, 1799, 488, 2693, 47, 6, 392, 173, 4, 2, 4378, 270, 2352, 4, 1500, 7, 4, 65, 55, 73, 11,
346, 14, 20, 9, 6, 976, 2078, 7, 5293, 861, 2, 5, 4182, 30, 3127, 2, 56, 4, 841, 5, 990, 692, 8, 4, 1669, 398, 229, 10,
10, 13, 2822, 670, 5304, 14, 9, 31, 7, 27, 111, 108, 15, 2033, 19, 7836, 1429, 875, 551, 14, 22, 9, 1193, 21, 45, 4829,
5, 45, 252, 8, 2, 6, 565, 921, 3639, 39, 4, 529, 48, 25, 181, 8, 67, 35, 1732, 22, 49, 238, 60, 135, 1162, 14, 9, 290,
4, 58, 10, 10, 472, 45, 55, 878, 8, 169, 11, 374, 5687, 25, 203, 28, 8, 818, 12, 125, 4, 3077]),
       list([1, 111, 748, 4368, 1133, 2, 2, 4, 87, 1551, 1262, 7, 31, 318, 9459, 7, 4, 498, 5076, 748, 63, 29, 5161,
220, 686, 2, 5, 17, 12, 575, 220, 2507, 17, 6, 185, 132, 2, 16, 53, 928, 11, 2, 74, 4, 438, 21, 27, 2, 589, 8, 22, 107,
2, 2, 997, 1638, 8, 35, 2076, 9019, 11, 22, 231, 54, 29, 1706, 29, 100, 2, 2425, 34, 2, 8738, 2, 5, 2, 98, 31, 2122,
33, 6, 58, 14, 3808, 1638, 8, 4, 365, 7, 2789, 3761, 356, 346, 4, 2, 1060, 63, 29, 93, 11, 5421, 11, 2, 33, 6, 58, 54,
1270, 431, 748, 7, 32, 2580, 16, 11, 94, 2, 10, 10, 4, 993, 2, 7, 4, 1766, 2634, 2164, 2, 8, 847, 8, 1450, 121, 31, 7,
27, 86, 2663, 2, 16, 6, 465, 993, 2006, 2, 573, 17, 2, 42, 4, 2, 37, 473, 6, 711, 6, 8869, 7, 328, 212, 70, 30, 258,
11, 220, 32, 7, 108, 21, 133, 12, 9, 55, 465, 849, 3711, 53, 33, 2071, 1969, 37, 70, 1144, 4, 5940, 1409, 74, 476, 37,
62, 91, 1329, 169, 4, 1330, 2, 146, 655, 2212, 5, 258, 12, 184, 2, 546, 5, 849, 2, 7, 4, 22, 1436, 18, 631, 1386, 797,
7, 4, 8712, 71, 348, 425, 4320, 1061, 19, 2, 5, 2, 11, 661, 8, 339, 2, 4, 2455, 2, 7, 4, 1962, 10, 10, 263, 787, 9,
270, 11, 6, 9466, 4, 2, 2, 121, 4, 5437, 26, 4434, 19, 68, 1372, 5, 28, 446, 6, 318, 7149, 8, 67, 51, 36, 70, 81, 8,
4392, 2294, 36, 1197, 8, 2, 2, 18, 6, 711, 4, 9909, 26, 2, 1125, 11, 14, 636, 720, 12, 426, 28, 77, 776, 8, 97, 38,
111, 7489, 6175, 168, 1239, 5189, 137, 2, 18, 27, 173, 9, 2399, 17, 6, 2, 428, 2, 232, 11, 4, 8014, 37, 272, 40, 2708,
247, 30, 656, 6, 2, 54, 2, 3292, 98, 6, 2840, 40, 558, 37, 6093, 98, 4, 2, 1197, 15, 14, 9, 57, 4893, 5, 4659, 6, 275,
711, 7937, 2, 3292, 98, 6, 2, 10, 10, 6639, 19, 14, 2, 267, 162, 711, 37, 5900, 752, 98, 4, 2, 2378, 90, 19, 6, 2, 7,
2, 1810, 2, 4, 4770, 3183, 930, 8, 508, 90, 4, 1317, 8, 4, 2, 17, 2, 3965, 1853, 4, 1494, 8, 4468, 189, 4, 2, 6287,
5774, 4, 4770, 5, 95, 271, 23, 6, 7742, 6063, 2, 5437, 33, 1526, 6, 425, 3155, 2, 4535, 1636, 7, 4, 4669, 2, 469, 4,
4552, 54, 4, 150, 5664, 2, 280, 53, 2, 2, 18, 339, 29, 1978, 27, 7885, 5, 2, 68, 1830, 19, 6571, 2, 4, 1515, 7, 263,
65, 2132, 34, 6, 5680, 7489, 43, 159, 29, 9, 4706, 9, 387, 73, 195, 584, 10, 10, 1069, 4, 58, 810, 54, 14, 6078, 117,
22, 16, 93, 5, 1069, 4, 192, 15, 12, 16, 93, 34, 6, 1766, 2, 33, 4, 5673, 7, 15, 2, 9252, 3286, 325, 12, 62, 30, 776,
8, 67, 14, 17, 6, 2, 44, 148, 687, 2, 203, 42, 203, 24, 28, 69, 2, 6676, 11, 330, 54, 29, 93, 2, 21, 845, 2, 27, 1099,
7, 819, 4, 22, 1407, 17, 6, 2, 787, 7, 2460, 2, 2, 100, 30, 4, 3737, 3617, 3169, 2321, 42, 1898, 11, 4, 3814, 42, 101,
704, 7, 101, 999, 15, 1625, 94, 2926, 180, 5, 9, 9101, 34, 2, 45, 6, 1429, 22, 60, 6, 1220, 31, 11, 94, 6408, 96, 21,
94, 749, 9, 57, 975]),
       ...,
       list([1, 13, 1408, 15, 8, 135, 14, 9, 35, 32, 46, 394, 20, 62, 30, 5093, 21, 45, 184, 78, 4, 1492, 910, 769,
2290, 2515, 395, 4257, 5, 1454, 11, 119, 2, 89, 1036, 4, 116, 218, 78, 21, 407, 100, 30, 128, 262, 15, 7, 185, 2280,
284, 1842, 2, 37, 315, 4, 226, 20, 272, 2942, 40, 29, 152, 60, 181, 8, 30, 50, 553, 362, 80, 119, 12, 21, 846, 5518]),
       list([1, 11, 119, 241, 9, 4, 840, 20, 12, 468, 15, 94, 3684, 562, 791, 39, 4, 86, 107, 8, 97, 14, 31, 33, 4,
2960, 7, 743, 46, 1028, 9, 3531, 5, 4, 768, 47, 8, 79, 90, 145, 164, 162, 50, 6, 501, 119, 7, 9, 4, 78, 232, 15, 16,
224, 11, 4, 333, 20, 4, 985, 200, 5, 2, 5, 9, 1861, 8, 79, 357, 4, 20, 47, 220, 57, 206, 139, 11, 12, 5, 55, 117, 212,
13, 1276, 92, 124, 51, 45, 1188, 71, 536, 13, 520, 14, 20, 6, 2302, 7, 470]),
       list([1, 6, 52, 7465, 430, 22, 9, 220, 2594, 8, 28, 2, 519, 3227, 6, 769, 15, 47, 6, 3482, 4067, 8, 114, 5, 33,
222, 31, 55, 184, 704, 5586, 2, 19, 346, 3153, 5, 6, 364, 350, 4, 184, 5586, 9, 133, 1810, 11, 5417, 2, 21, 4, 7298, 2,
570, 50, 2005, 2643, 9, 6, 1249, 17, 6, 2, 2, 21, 17, 6, 1211, 232, 1138, 2249, 29, 266, 56, 96, 346, 194, 308, 9, 194,
21, 29, 218, 1078, 19, 4, 78, 173, 7, 27, 2, 5698, 3406, 718, 2, 9, 6, 6907, 17, 210, 5, 3281, 5677, 47, 77, 395, 14,
172, 173, 18, 2740, 2931, 4517, 82, 127, 27, 173, 11, 6, 392, 217, 21, 50, 9, 57, 65, 12, 2, 53, 40, 35, 390, 7, 11, 4,
3567, 7, 4, 314, 74, 6, 792, 22, 2, 19, 714, 727, 5205, 382, 4, 91, 6533, 439, 19, 14, 20, 9, 1441, 5805, 1118, 4, 756,
25, 124, 4, 31, 12, 16, 93, 804, 34, 2005, 2643])],
      dtype=object)
```

```
te_labels[0]
```

```
0
```

```
max([max(seq) for seq in te_set])
```

```
9999
```

** Reviews to text**

```
word_to_index = imdb.get_word_index()
index_to_word_map = dict(
    [(value, key) for (key, value) in word_to_index.items()])
review_text = " ".join(
    [index_to_word_map.get(i - 3, "?") for i in tr_set[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
review_text
```

```
'? this film was just brilliant casting location scenery story direction everyone's really suited the part they played
and you could just imagine being there robert ? is an amazing actor and now the same being director ? father came from
the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks thr
oughout the film were great it was just brilliant so much that i bought the film as soon as it was released for ? and w
ould recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you kn
ow what they say if you cry at a film it must have been good and this definitely was also ? to the two little boy's tha
t played the ? of norman and paul they were just brilliant children are often left out of the ? list i think because th
e stars that play them all grown up are such a big profile for the whole film but these children are amazing and should
```

Data preparation

```
import numpy as np
def transform_input_sequences(input_sequences, vocab_size=10000):
    binary_matrix = np.zeros((len(input_sequences), vocab_size))
    for i, sequence in enumerate(input_sequences):
        for j in sequence:
            binary_matrix[i, j] = 1.
    return binary_matrix
```

Data Vectorization

```
train_dataset_1 = transform_input_sequences(tr_set)
test_dataset_1 = transform_input_sequences(te_set)
```

```
train_dataset_1[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

```
test_dataset_1[0]
```

```
array([0., 1., 1., ..., 0., 0., 0.])
```

Label Vectorization

```
train_dataset_2 = np.asarray(tr_labels).astype("float32")
test_data_2 = np.asarray(te_labels).astype("float32")
```

Building model using relu and compiling it

```
from tensorflow import keras
from tensorflow.keras import layers
seed(123)
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

```
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

```
seed(123)
x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]
y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]
```

```
seed(123)
history = model.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - accuracy: 0.6972 - loss: 0.5965 - val_accuracy: 0.8536 - val_loss: 0.3964
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8935 - loss: 0.3332 - val_accuracy: 0.8841 - val_loss: 0.3086
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9250 - loss: 0.2390 - val_accuracy: 0.8892 - val_loss: 0.2833
```

```
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9394 - loss: 0.1891 - val_accuracy: 0.8767 - val_loss: 0.3058
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9462 - loss: 0.1613 - val_accuracy: 0.8886 - val_loss: 0.2852
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9585 - loss: 0.1343 - val_accuracy: 0.8836 - val_loss: 0.2886
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9665 - loss: 0.1156 - val_accuracy: 0.8759 - val_loss: 0.3176
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9733 - loss: 0.0973 - val_accuracy: 0.8839 - val_loss: 0.3090
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9777 - loss: 0.0841 - val_accuracy: 0.8808 - val_loss: 0.3476
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9801 - loss: 0.0764 - val_accuracy: 0.8821 - val_loss: 0.3431
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9852 - loss: 0.0616 - val_accuracy: 0.8796 - val_loss: 0.3646
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9881 - loss: 0.0542 - val_accuracy: 0.8740 - val_loss: 0.3837
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9905 - loss: 0.0443 - val_accuracy: 0.8769 - val_loss: 0.4142
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9922 - loss: 0.0408 - val_accuracy: 0.8749 - val_loss: 0.4263
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9937 - loss: 0.0349 - val_accuracy: 0.8558 - val_loss: 0.5149
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9943 - loss: 0.0307 - val_accuracy: 0.8753 - val_loss: 0.4736
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9975 - loss: 0.0215 - val_accuracy: 0.8710 - val_loss: 0.4916
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9988 - loss: 0.0178 - val_accuracy: 0.8592 - val_loss: 0.5589
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9974 - loss: 0.0212 - val_accuracy: 0.8685 - val_loss: 0.5467
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9996 - loss: 0.0119 - val_accuracy: 0.8564 - val_loss: 0.6180
```

The initial training phase resulted in a loss of 0.5965 with an accuracy of 69.72% on the training data, while the validation data showed a loss of 0.3964 and an accuracy of 85.36%.

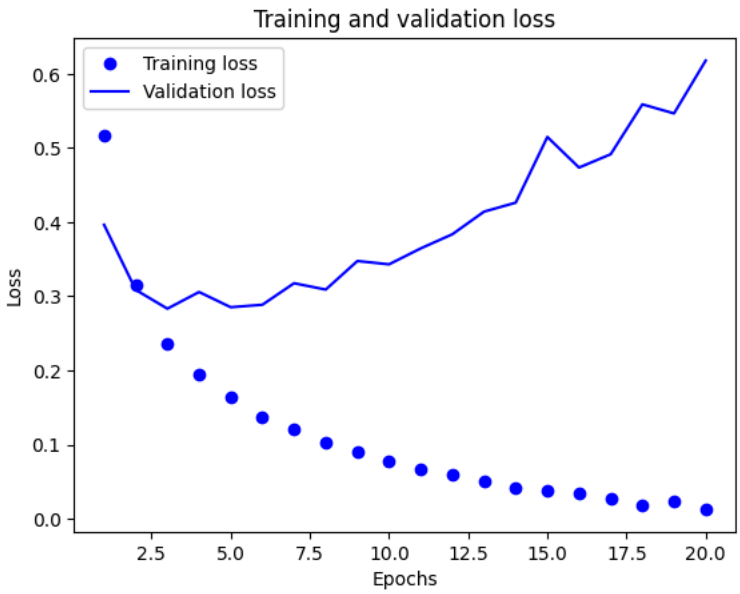
As training progressed, the model's accuracy on the training data steadily improved, reaching 99.96% with a loss of 0.0119 by Epoch 20. However, the validation accuracy declined to 85.64%, with an increased loss of 0.6180, indicating overfitting. The model demonstrated strong learning on the training set but struggled to generalize effectively to new data.

```
history__data = history.history
history__data.keys()
```

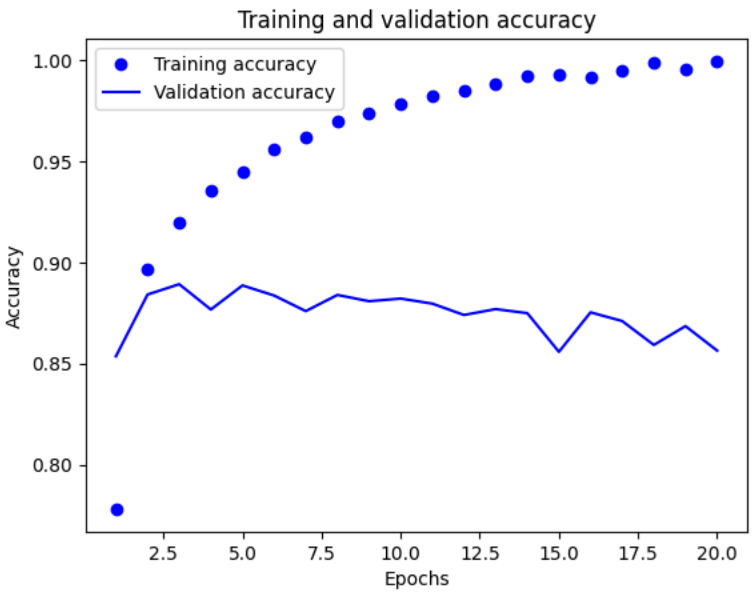
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

Plotting the training and validation loss

```
import matplotlib.pyplot as plt
history__data = history.history
loss_values = history__data["loss"]
val_loss_values = history__data["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history__data["accuracy"]
val_acc = history__data["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training accuracy")
plt.plot(epochs, val_acc, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



The graphs indicate overfitting, as training loss continuously decreases while validation loss rises after a few epochs. Despite achieving near 100% training accuracy, the fluctuating validation accuracy suggests poor generalization, requiring techniques like regularization or early stopping to improve performance.

Retraining the model

```
np.random.seed(123)
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

```
model.fit(train_dataset_1, train_dataset_2, epochs=4, batch_size=512)
binary_matrix = model.evaluate(test_dataset_1, test_data_2)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.7353 - loss: 0.5362
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9105 - loss: 0.2576
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9331 - loss: 0.1979
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9405 - loss: 0.1678
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8807 - loss: 0.2969
```

```
binary_matrix
```

```
[0.2942996621131897, 0.8832799792289734]
```

During testing of the neural network model it reached 88.32% accuracy with 0.2942 loss. The model applies successfully to new data points despite continuing signs of overfitting from its training results.

```
model.predict(test_dataset_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.18455303],
       [0.9998895 ],
       [0.8151574 ],
       ...,
       [0.10875935],
       [0.06385516],
       [0.70870835]], dtype=float32)
```

Building a neural network with 1 hidden layer

```
seed(123)
model1 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model1.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]

y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]


history1 = model1.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

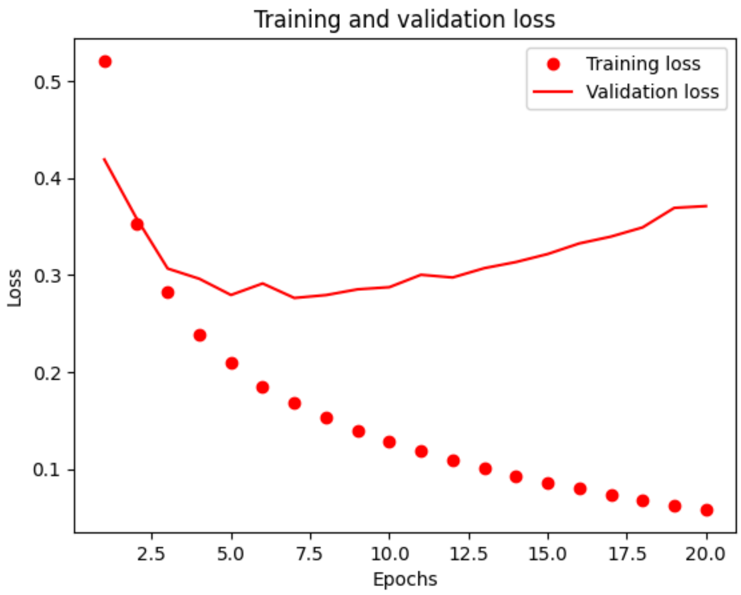
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 0.7170 - loss: 0.5882 - val_accuracy: 0.8568 - val_loss: 0.4191
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.8828 - loss: 0.3736 - val_accuracy: 0.8650 - val_loss: 0.3594
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9085 - loss: 0.2890 - val_accuracy: 0.8860 - val_loss: 0.3067
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9231 - loss: 0.2426 - val_accuracy: 0.8829 - val_loss: 0.2961
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9330 - loss: 0.2093 - val_accuracy: 0.8892 - val_loss: 0.2794
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9443 - loss: 0.1850 - val_accuracy: 0.8822 - val_loss: 0.2913
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9490 - loss: 0.1661 - val_accuracy: 0.8885 - val_loss: 0.2764
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9565 - loss: 0.1484 - val_accuracy: 0.8851 - val_loss: 0.2793
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9608 - loss: 0.1402 - val_accuracy: 0.8838 - val_loss: 0.2853
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9620 - loss: 0.1286 - val_accuracy: 0.8864 - val_loss: 0.2874
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9684 - loss: 0.1143 - val_accuracy: 0.8817 - val_loss: 0.3002
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9715 - loss: 0.1065 - val_accuracy: 0.8839 - val_loss: 0.2975
Epoch 13/20
```

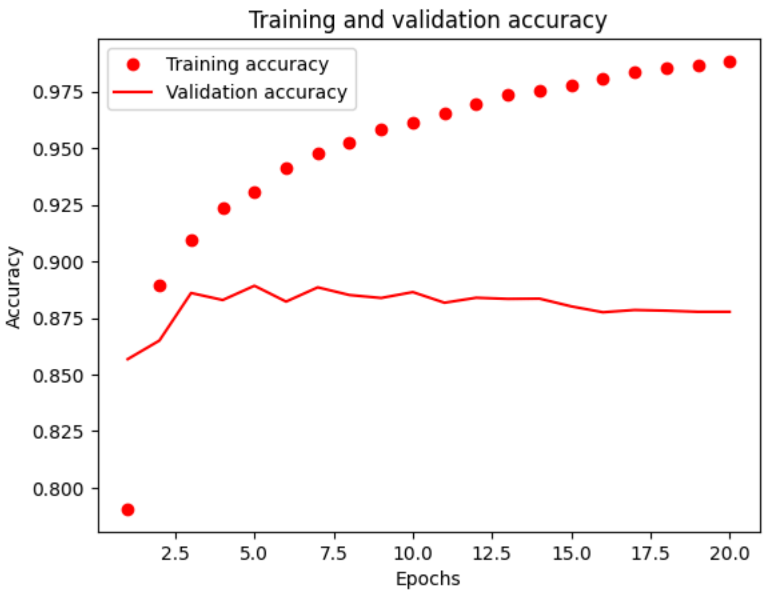
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9752 - loss: 0.0987 - val_accuracy: 0.8834 - val_loss: 0.3071
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9751 - loss: 0.0955 - val_accuracy: 0.8835 - val_loss: 0.3134
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9811 - loss: 0.0819 - val_accuracy: 0.8801 - val_loss: 0.3216
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9820 - loss: 0.0789 - val_accuracy: 0.8775 - val_loss: 0.3325
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9854 - loss: 0.0706 - val_accuracy: 0.8785 - val_loss: 0.3396
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9875 - loss: 0.0655 - val_accuracy: 0.8782 - val_loss: 0.3491
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9883 - loss: 0.0606 - val_accuracy: 0.8777 - val_loss: 0.3692
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9891 - loss: 0.0581 - val_accuracy: 0.8777 - val_loss: 0.3710
```

```
history_dict = history1.history
history_dict.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
import matplotlib.pyplot as plt
history_dict = history1.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
#Plotting graph between Training and Validation loss
plt.plot(epochs, loss_values, "ro", label="Training loss")
plt.plot(epochs, val_loss_values, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

#Plotting graph between Training and Validation Accuracy
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["val_accuracy"]
plt.plot(epochs, acc, "ro", label="Training accuracy")
plt.plot(epochs, val_acc, "r", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```






```
np.random.seed(123)
model1 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model1.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model1.fit(train_dataset_1, train_dataset_2, epochs=5, batch_size=512)
binary_matrix1 = model1.evaluate(test_dataset_1, test_data_2)
```

```
Epoch 1/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7498 - loss: 0.5409
Epoch 2/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.8973 - loss: 0.3006
Epoch 3/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9206 - loss: 0.2333
Epoch 4/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9295 - loss: 0.2037
Epoch 5/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9401 - loss: 0.1801
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8855 - loss: 0.2820
```

```
binary_matrix1
```

```
[0.2800644040107727, 0.8880400061607361]
```

The test set achieved a loss of 0.280 and an accuracy of 88.80%.

```
model1.predict(test_dataset_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.24095112],
       [0.99988824],
       [0.8748426 ],
       ...,
       [0.11667909],
       [0.09097715],
       [0.63858324]], dtype=float32)
```

Creating a neural network with three hidden layers

```
np.random.seed(123)
model_3 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_3.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]

y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]

history3 = model_3.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

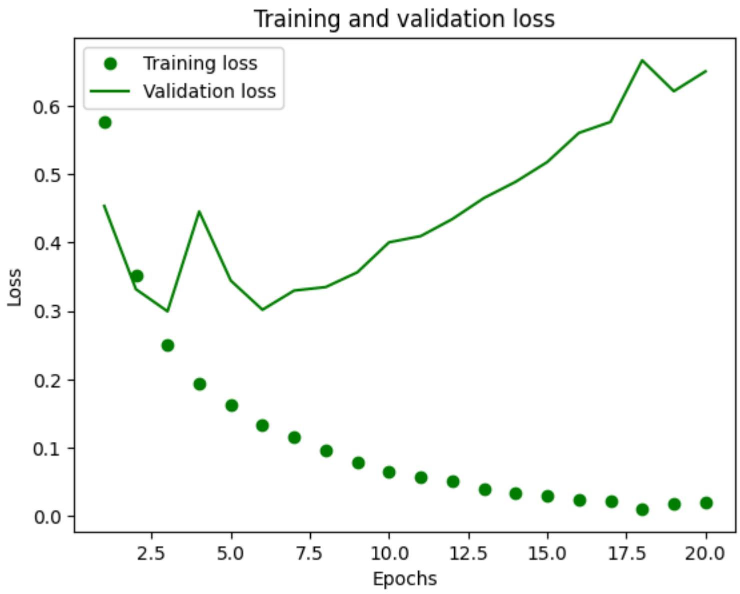
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.6840 - loss: 0.6369 - val_accuracy: 0.8239 - val_loss: 0.4534
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - accuracy: 0.8816 - loss: 0.3782 - val_accuracy: 0.8764 - val_loss: 0.3317
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9195 - loss: 0.2575 - val_accuracy: 0.8835 - val_loss: 0.2993
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9419 - loss: 0.1914 - val_accuracy: 0.8263 - val_loss: 0.4453
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9450 - loss: 0.1720 - val_accuracy: 0.8693 - val_loss: 0.3442
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9592 - loss: 0.1347 - val_accuracy: 0.8846 - val_loss: 0.3016
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9676 - loss: 0.1074 - val_accuracy: 0.8818 - val_loss: 0.3297
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9747 - loss: 0.0885 - val_accuracy: 0.8789 - val_loss: 0.3348
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9804 - loss: 0.0766 - val_accuracy: 0.8805 - val_loss: 0.3564
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9840 - loss: 0.0620 - val_accuracy: 0.8720 - val_loss: 0.4002
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9858 - loss: 0.0574 - val_accuracy: 0.8762 - val_loss: 0.4093
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9893 - loss: 0.0460 - val_accuracy: 0.8752 - val_loss: 0.4342
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9936 - loss: 0.0343 - val_accuracy: 0.8747 - val_loss: 0.4650
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9940 - loss: 0.0293 - val_accuracy: 0.8739 - val_loss: 0.4886
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9961 - loss: 0.0225 - val_accuracy: 0.8721 - val_loss: 0.5176
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9972 - loss: 0.0185 - val_accuracy: 0.8715 - val_loss: 0.5602
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9975 - loss: 0.0147 - val_accuracy: 0.8710 - val_loss: 0.5762
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9986 - loss: 0.0123 - val_accuracy: 0.8689 - val_loss: 0.6663
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9931 - loss: 0.0259 - val_accuracy: 0.8679 - val_loss: 0.6211
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9982 - loss: 0.0102 - val_accuracy: 0.8682 - val_loss: 0.6501
```

```
history_dict3 = history3.history
history_dict3.keys()
```

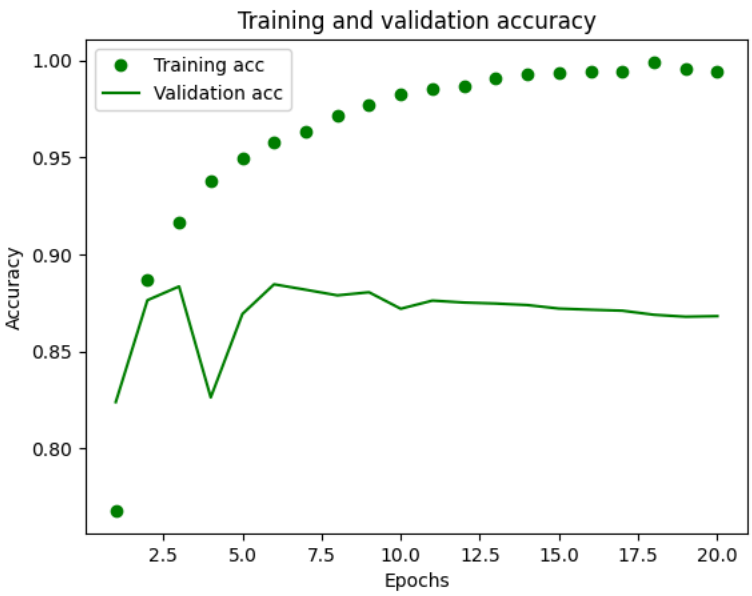
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = history_dict3["loss"]
val_loss_values = history_dict3["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "go", label="Training loss")
```

```
plt.plot(epochs, val_loss_values, "g", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict3["accuracy"]
val_acc = history_dict3["val_accuracy"]
plt.plot(epochs, acc, "go", label="Training acc")
plt.plot(epochs, val_acc, "g", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
np.random.seed(123)
model_3 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

```
model_3.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
model_3.fit(train_dataset_1, train_dataset_2, epochs=3, batch_size=512)
binary_matrix_3 = model_3.evaluate(test_dataset_1, test_data_2)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.7149 - loss: 0.5935
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.8970 - loss: 0.3017
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9252 - loss: 0.2189
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8836 - loss: 0.2916
```

The test set has a loss of 0.29 and an accuracy of 88.36%.

```
binary_matrix_3
```

```
[0.2881566286087036, 0.8846399784088135]
```

```
model_3.predict(test_dataset_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.29381514],
       [0.9993426 ],
       [0.9373727 ],
       ...,
       [0.15358119],
       [0.12072782],
       [0.6303163 ]], dtype=float32)
```

Building Neural Network with 32 units.

```
np.random.seed(123)
model_32 = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
#model compilation
model_32.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
#model validation
x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]

y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]

np.random.seed(123)
history32 = model_32.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

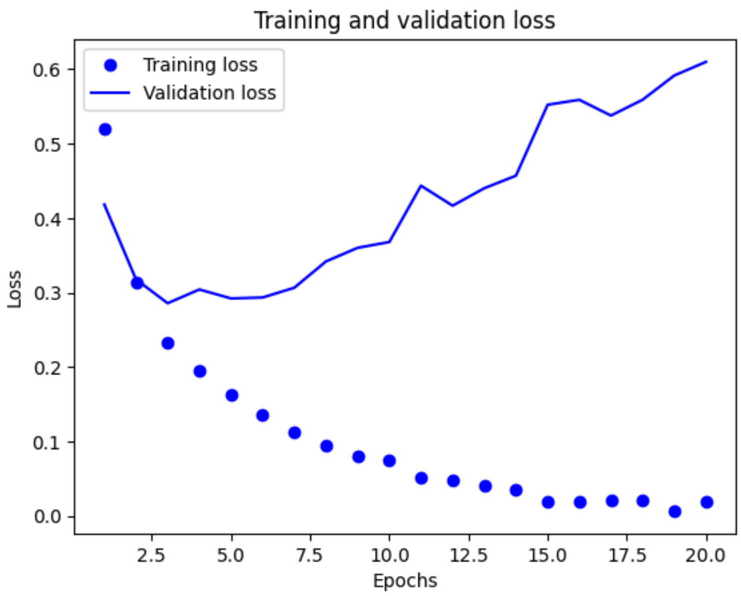
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.6914 - loss: 0.5997 - val_accuracy: 0.8241 - val_loss: 0.4183
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.8871 - loss: 0.3291 - val_accuracy: 0.8771 - val_loss: 0.3177
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9173 - loss: 0.2379 - val_accuracy: 0.8863 - val_loss: 0.2857
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9324 - loss: 0.1975 - val_accuracy: 0.8792 - val_loss: 0.3042
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9490 - loss: 0.1601 - val_accuracy: 0.8821 - val_loss: 0.2922
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9595 - loss: 0.1288 - val_accuracy: 0.8847 - val_loss: 0.2934
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9657 - loss: 0.1106 - val_accuracy: 0.8840 - val_loss: 0.3065
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9726 - loss: 0.0924 - val_accuracy: 0.8740 - val_loss: 0.3419
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9767 - loss: 0.0792 - val_accuracy: 0.8795 - val_loss: 0.3601
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9800 - loss: 0.0722 - val_accuracy: 0.8759 - val_loss: 0.3679
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9896 - loss: 0.0479 - val_accuracy: 0.8698 - val_loss: 0.4436
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9896 - loss: 0.0450 - val_accuracy: 0.8723 - val_loss: 0.4167
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9939 - loss: 0.0348 - val_accuracy: 0.8720 - val_loss: 0.4402
Epoch 14/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9970 - loss: 0.0242 - val_accuracy: 0.8752 - val_loss: 0.4570
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9987 - loss: 0.0170 - val_accuracy: 0.8569 - val_loss: 0.5523
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9931 - loss: 0.0281 - val_accuracy: 0.8679 - val_loss: 0.5589
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9884 - loss: 0.0333 - val_accuracy: 0.8751 - val_loss: 0.5378
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9968 - loss: 0.0158 - val_accuracy: 0.8710 - val_loss: 0.5590
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9998 - loss: 0.0066 - val_accuracy: 0.8694 - val_loss: 0.5916
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9962 - loss: 0.0141 - val_accuracy: 0.8715 - val_loss: 0.6099
```

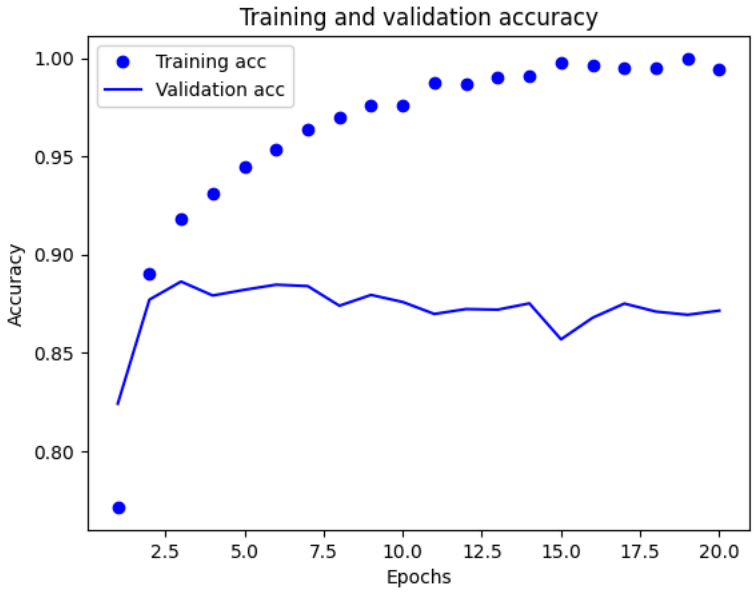
```
hyper_model_history_dict = history32.history
hyper_model_history_dict.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = hyper_model_history_dict["loss"]
val_loss_values = hyper_model_history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = hyper_model_history_dict["accuracy"]
val_acc = hyper_model_history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
history_32 = model_32.fit(train_dataset_1, train_dataset_2, epochs=3, batch_size=12)
final_result_32 = model_32.evaluate(test_dataset_1, test_data_2)
print(final_result_32)
```

```
Epoch 1/3
2084/2084 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9316 - loss: 0.2236
Epoch 2/3
2084/2084 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - accuracy: 0.9411 - loss: 0.1628
Epoch 3/3
2084/2084 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9544 - loss: 0.1357
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8695 - loss: 0.4482
[0.4485381543636322, 0.8711199760437012]
```

```
model_32.predict(test_dataset_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.04204874],
       [1.        ],
       [0.6628257 ],
       ...,
       [0.03195171],
       [0.01308174],
       [0.24813423]], dtype=float32)
```

Training the model with 64 units

```
np.random.seed(123)
model_64 = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_64.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]

y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]

np.random.seed(123)
history64 = model_64.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

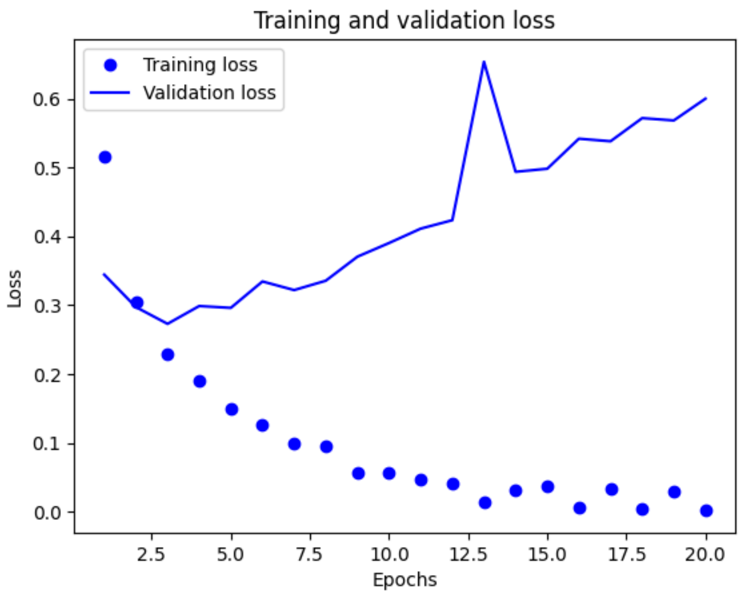
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - accuracy: 0.6616 - loss: 0.5971 - val_accuracy: 0.8699 - val_loss: 0.3443
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.8741 - loss: 0.3211 - val_accuracy: 0.8793 - val_loss: 0.2973
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 105ms/step - accuracy: 0.9137 - loss: 0.2288 - val_accuracy: 0.8893 - val_loss: 0.2729
Epoch 4/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.9318 - loss: 0.1806 - val_accuracy: 0.8803 - val_loss: 0.2987
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9487 - loss: 0.1450 - val_accuracy: 0.8850 - val_loss: 0.2961
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9616 - loss: 0.1136 - val_accuracy: 0.8781 - val_loss: 0.3345
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9719 - loss: 0.0886 - val_accuracy: 0.8842 - val_loss: 0.3219
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9768 - loss: 0.0737 - val_accuracy: 0.8829 - val_loss: 0.3355
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9880 - loss: 0.0519 - val_accuracy: 0.8786 - val_loss: 0.3704
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - accuracy: 0.9886 - loss: 0.0453 - val_accuracy: 0.8806 - val_loss: 0.3903
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9912 - loss: 0.0358 - val_accuracy: 0.8780 - val_loss: 0.4111
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9960 - loss: 0.0225 - val_accuracy: 0.8785 - val_loss: 0.4234
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9993 - loss: 0.0128 - val_accuracy: 0.8418 - val_loss: 0.6535
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9742 - loss: 0.0679 - val_accuracy: 0.8756 - val_loss: 0.4937
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9936 - loss: 0.0235 - val_accuracy: 0.8766 - val_loss: 0.4981
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9999 - loss: 0.0057 - val_accuracy: 0.8765 - val_loss: 0.5418
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9955 - loss: 0.0167 - val_accuracy: 0.8751 - val_loss: 0.5380
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 1.0000 - loss: 0.0036 - val_accuracy: 0.8759 - val_loss: 0.5716
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9980 - loss: 0.0086 - val_accuracy: 0.8746 - val_loss: 0.5683
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 1.0000 - loss: 0.0027 - val_accuracy: 0.8750 - val_loss: 0.5999
```

```
history_dict64 = history64.history
history_dict64.keys()
```

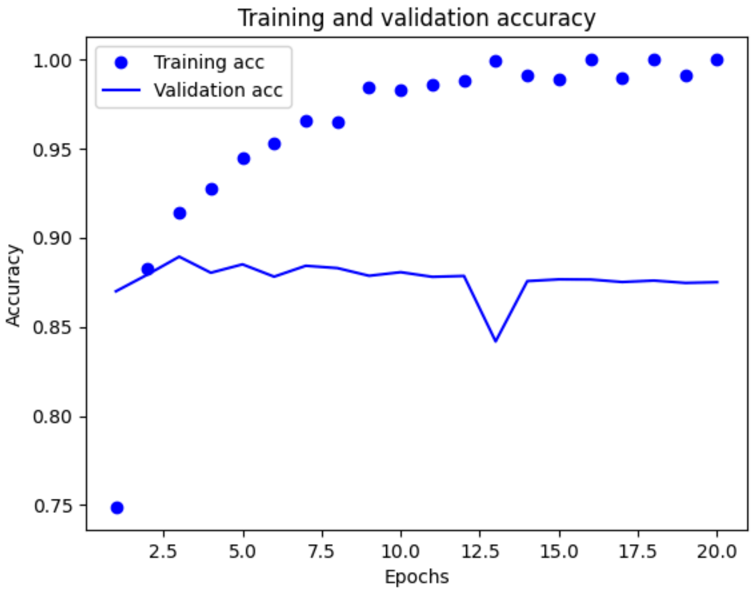
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = history_dict64["loss"]
val_loss_values = history_dict64["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict64["accuracy"]
val_acc = history_dict64["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
```

```
plt.legend()
plt.show()
```




```
history_64 = model_64.fit(train_dataset_1, train_dataset_2, epochs=3, batch_size=512)
binary_matrix_64 = model_64.evaluate(test_dataset_1, test_data_2)
binary_matrix_64
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9434 - loss: 0.2062
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9731 - loss: 0.0895
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.9850 - loss: 0.0558
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8684 - loss: 0.4019
[0.4005260467529297, 0.8715199828147888]
```

```
model_64.predict(test_dataset_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
array([[0.0133222 ],
       [0.9999998 ],
       [0.49194503],
       ...,
       [0.01720217],
       [0.01750094],
       [0.9095838 ]], dtype=float32)
```

The test set, the accuracy dropped to 87.15% with a higher loss of 0.4005

Training the model with 128 units

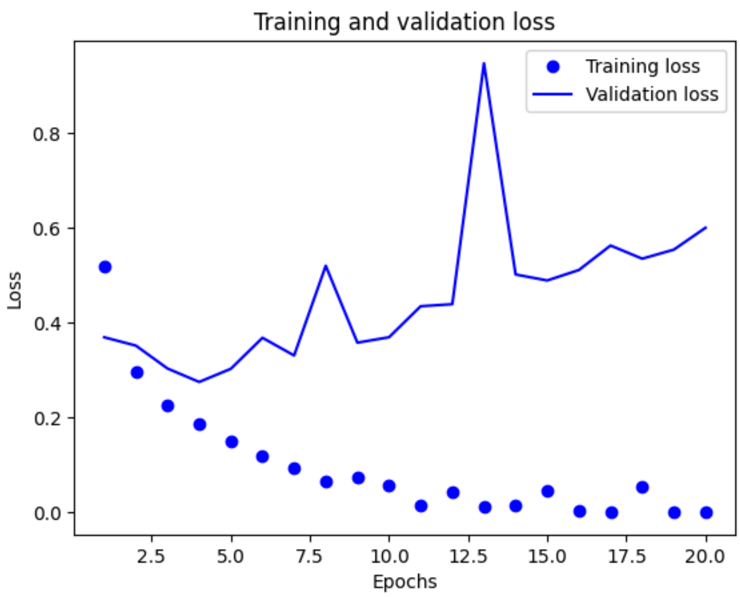
```
np.random.seed(123)
model_128 = keras.Sequential([
    layers.Dense(128, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_128.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]

y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]

np.random.seed(123)
history128 = model_128.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 85ms/step - accuracy: 0.6779 - loss: 0.5957 - val_accuracy: 0.8525 - val_loss: 0.3694
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 65ms/step - accuracy: 0.8862 - loss: 0.3044 - val_accuracy: 0.8477 - val_loss: 0.3518
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 81ms/step - accuracy: 0.9106 - loss: 0.2322 - val_accuracy: 0.8780 - val_loss: 0.3035
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.9385 - loss: 0.1730 - val_accuracy: 0.8861 - val_loss: 0.2752
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9578 - loss: 0.1296 - val_accuracy: 0.8819 - val_loss: 0.3029
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - accuracy: 0.9679 - loss: 0.1046 - val_accuracy: 0.8640 - val_loss: 0.3683
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9740 - loss: 0.0868 - val_accuracy: 0.8816 - val_loss: 0.3310
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 78ms/step - accuracy: 0.9836 - loss: 0.0590 - val_accuracy: 0.8492 - val_loss: 0.5201
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9667 - loss: 0.0857 - val_accuracy: 0.8783 - val_loss: 0.3580
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9955 - loss: 0.0267 - val_accuracy: 0.8808 - val_loss: 0.3694
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.9988 - loss: 0.0156 - val_accuracy: 0.8792 - val_loss: 0.4348
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9843 - loss: 0.0457 - val_accuracy: 0.8783 - val_loss: 0.4392
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step - accuracy: 0.9997 - loss: 0.0069 - val_accuracy: 0.8044 - val_loss: 0.9469
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 66ms/step - accuracy: 0.9853 - loss: 0.0363 - val_accuracy: 0.8778 - val_loss: 0.5017
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 66ms/step - accuracy: 0.9988 - loss: 0.0069 - val_accuracy: 0.8764 - val_loss: 0.4891
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 1.0000 - loss: 0.0053 - val_accuracy: 0.8768 - val_loss: 0.5112
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 0.8767 - val_loss: 0.5628
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.9968 - loss: 0.0136 - val_accuracy: 0.8765 - val_loss: 0.5351
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.8794 - val_loss: 0.5541
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.8780 - val_loss: 0.6002
```

```
history_dict128 = history128.history
history_dict128.keys()
```

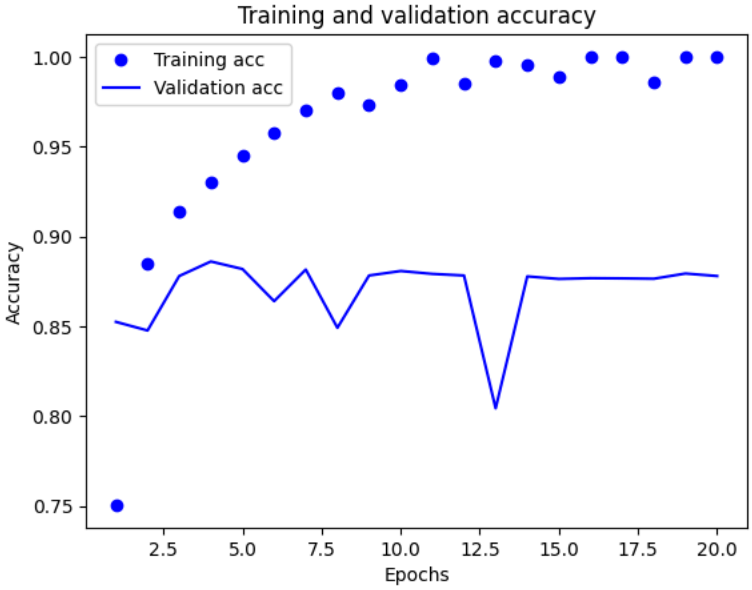
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = history_dict128["loss"]
val_loss_values = history_dict128["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict128["accuracy"]
```

```
val_acc = history_dict128["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
history_128 = model_128.fit(train_dataset_1, train_dataset_2, epochs=2, batch_size=512)
binary_matrix_128 = model_128.evaluate(test_dataset_1, test_data_2)
binary_matrix_128
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.9381 - loss: 0.2200
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9775 - loss: 0.0783
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8657 - loss: 0.3741
[0.37036311626434326, 0.8686800003051758]
```

```
model_128.predict(test_dataset_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
array([[0.01722546],
       [0.9999987 ],
       [0.7190469 ],
       ...,
       [0.10129648],
       [0.01521256],
       [0.95121455]], dtype=float32)
```

The validation set has an accuracy of 86.86%.

MSE Loss Function

```
np.random.seed(123)
model_MSE = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
#Model compilation
model_MSE.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
# validation
x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]

y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]
# Model Fit
np.random.seed(123)
history_model_MSE = model_MSE.fit(partial_train_dataset_1,
```

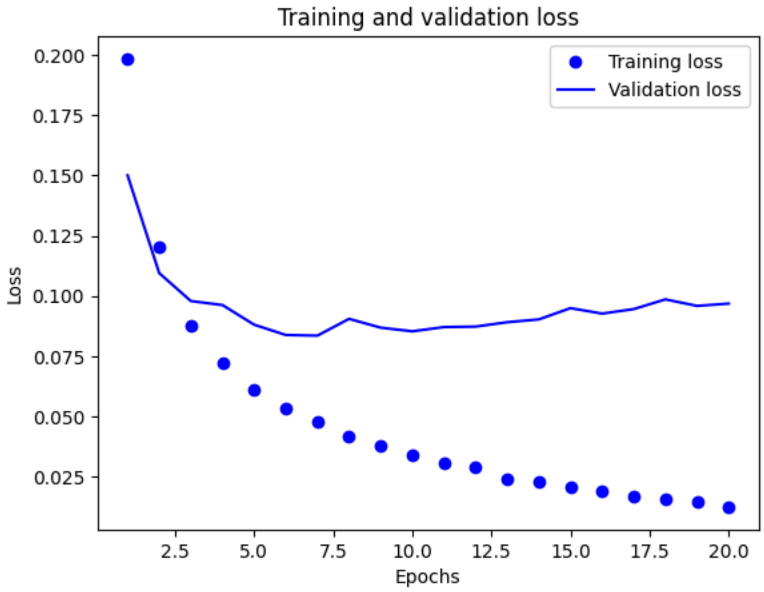
```
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - accuracy: 0.6701 - loss: 0.2236 - val_accuracy: 0.8344 - val_loss: 0.1500
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8743 - loss: 0.1285 - val_accuracy: 0.8778 - val_loss: 0.1095
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9057 - loss: 0.0915 - val_accuracy: 0.8769 - val_loss: 0.0978
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9200 - loss: 0.0726 - val_accuracy: 0.8733 - val_loss: 0.0962
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9319 - loss: 0.0618 - val_accuracy: 0.8822 - val_loss: 0.0880
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9424 - loss: 0.0541 - val_accuracy: 0.8873 - val_loss: 0.0838
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9509 - loss: 0.0473 - val_accuracy: 0.8857 - val_loss: 0.0835
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9615 - loss: 0.0402 - val_accuracy: 0.8744 - val_loss: 0.0905
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9646 - loss: 0.0370 - val_accuracy: 0.8777 - val_loss: 0.0868
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9701 - loss: 0.0314 - val_accuracy: 0.8825 - val_loss: 0.0853
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9720 - loss: 0.0304 - val_accuracy: 0.8823 - val_loss: 0.0870
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9729 - loss: 0.0275 - val_accuracy: 0.8799 - val_loss: 0.0872
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9811 - loss: 0.0229 - val_accuracy: 0.8774 - val_loss: 0.0891
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9838 - loss: 0.0204 - val_accuracy: 0.8782 - val_loss: 0.0902
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9842 - loss: 0.0201 - val_accuracy: 0.8771 - val_loss: 0.0949
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9844 - loss: 0.0192 - val_accuracy: 0.8747 - val_loss: 0.0926
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9886 - loss: 0.0158 - val_accuracy: 0.8771 - val_loss: 0.0945
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9885 - loss: 0.0154 - val_accuracy: 0.8749 - val_loss: 0.0985
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9906 - loss: 0.0136 - val_accuracy: 0.8736 - val_loss: 0.0958
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9927 - loss: 0.0110 - val_accuracy: 0.8728 - val_loss: 0.0968
```

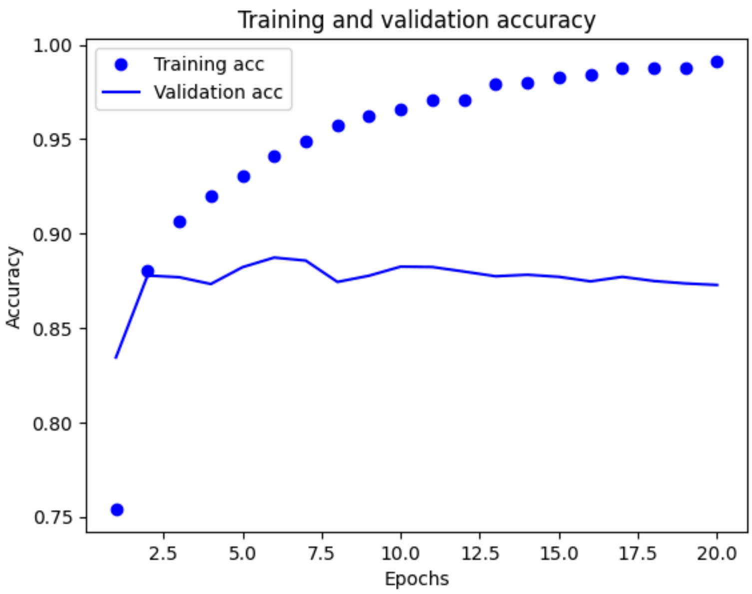
```
history_dict_MSE = history_model_MSE.history
history_dict_MSE.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
import matplotlib.pyplot as plt
loss_values = history_dict_MSE["loss"]
val_loss_values = history_dict_MSE["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict_MSE["accuracy"]
val_acc = history_dict_MSE["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
model_MSE.fit(train_dataset_1, train_dataset_2, epochs=8, batch_size=512)
binary_mse = model_MSE.evaluate(test_dataset_1, test_data_2)
binary_mse
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9437 - loss: 0.0476
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9574 - loss: 0.0376
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9678 - loss: 0.0305
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9693 - loss: 0.0292
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9761 - loss: 0.0244
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9765 - loss: 0.0244
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9784 - loss: 0.0226
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9786 - loss: 0.0221
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8624 - loss: 0.1107
```

```
[0.10755906254053116, 0.8662800192832947]
```

```
model_MSE.predict(test_dataset_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.03596858],
       [1.        ],
       [0.9269522 ],
       ...,
       [0.09182716],
       [0.02129245],
       [0.96913606]], dtype=float32)
```

Tanh Activation Function

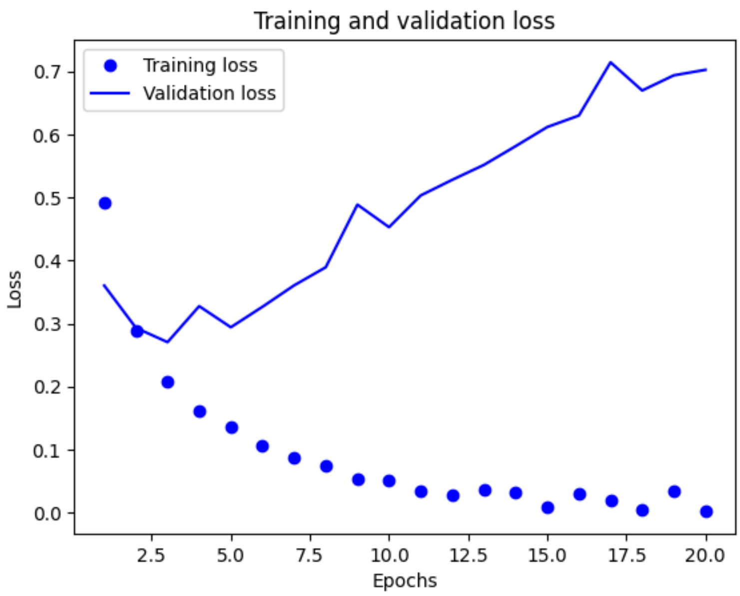
```
np.random.seed(123)
model_tanh = keras.Sequential([
    layers.Dense(16, activation="tanh"),
    layers.Dense(16, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])

model_tanh.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]

y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]

np.random.seed(123)

history_tanh = model_tanh.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.6964 - loss: 0.5717 - val_accuracy: 0.8725 - val_loss: 0.3602
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9013 - loss: 0.3001 - val_accuracy: 0.8862 - val_loss: 0.2931
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9283 - loss: 0.2094 - val_accuracy: 0.8899 - val_loss: 0.2703
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 78ms/step - accuracy: 0.9516 - loss: 0.1565 - val_accuracy: 0.8687 - val_loss: 0.3275
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9555 - loss: 0.1276 - val_accuracy: 0.8853 - val_loss: 0.2940
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9688 - loss: 0.1006 - val_accuracy: 0.8841 - val_loss: 0.3265
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9752 - loss: 0.0783 - val_accuracy: 0.8753 - val_loss: 0.3605
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9792 - loss: 0.0689 - val_accuracy: 0.8732 - val_loss: 0.3892
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9868 - loss: 0.0508 - val_accuracy: 0.8639 - val_loss: 0.4886
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9903 - loss: 0.0368 - val_accuracy: 0.8744 - val_loss: 0.4528
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9938 - loss: 0.0258 - val_accuracy: 0.8665 - val_loss: 0.5031
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9940 - loss: 0.0254 - val_accuracy: 0.8702 - val_loss: 0.5280
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9910 - loss: 0.0326 - val_accuracy: 0.8706 - val_loss: 0.5517
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9975 - loss: 0.0143 - val_accuracy: 0.8693 - val_loss: 0.5814
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9989 - loss: 0.0090 - val_accuracy: 0.8664 - val_loss: 0.6118
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9927 - loss: 0.0250 - val_accuracy: 0.8683 - val_loss: 0.6300
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9994 - loss: 0.0055 - val_accuracy: 0.8605 - val_loss: 0.7144
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9980 - loss: 0.0077 - val_accuracy: 0.8670 - val_loss: 0.6697
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9989 - loss: 0.0059 - val_accuracy: 0.8663 - val_loss: 0.6938
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9999 - loss: 0.0025 - val_accuracy: 0.8660 - val_loss: 0.7025
```

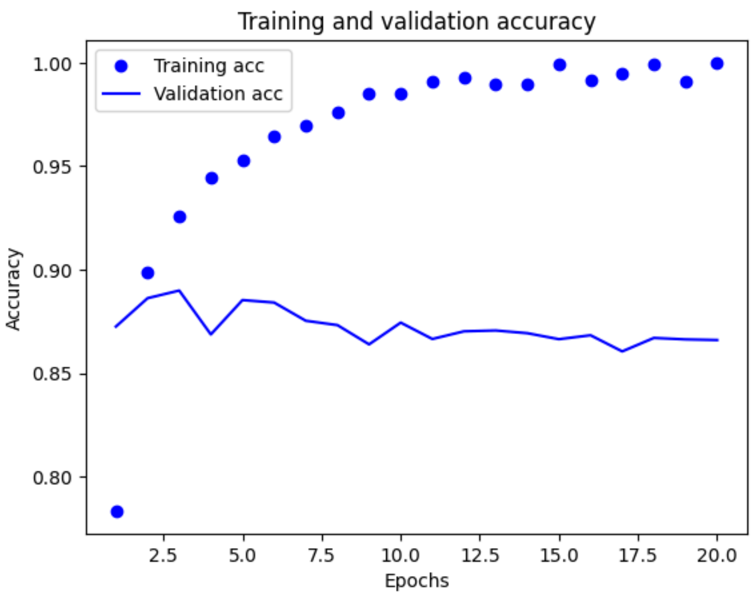
```
history_dict_tanh = history_tanh.history
history_dict_tanh.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = history_dict_tanh["loss"]
val_loss_values = history_dict_tanh["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict_tanh["accuracy"]
val_acc = history_dict_tanh["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
model_tanh.fit(train_dataset_1, train_dataset_2, epochs=8, batch_size=512)
binary_matrix_tanh = model_tanh.evaluate(test_dataset_1, test_data_2)
binary_matrix_tanh
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9411 - loss: 0.2894
```

```
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9604 - loss: 0.1434
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9682 - loss: 0.1070
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9721 - loss: 0.0897
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9773 - loss: 0.0742
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9769 - loss: 0.0705
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9850 - loss: 0.0569
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9789 - loss: 0.0665
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8535 - loss: 0.6079
[0.599612832069397, 0.8537600040435791]
```

Adam Optimizer Function

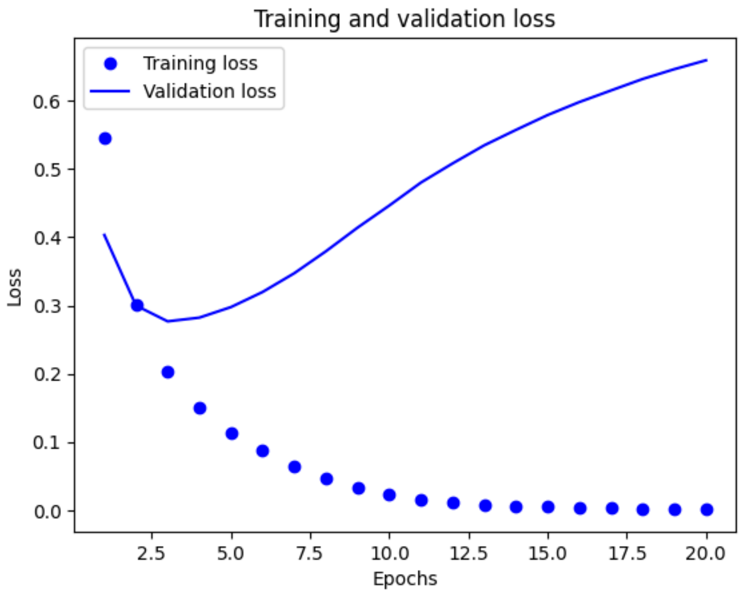
```
np.random.seed(123)
model_adam = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model_adam.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

x_validation = train_dataset_1[:10000]
partial_train_dataset_1 = train_dataset_1[10000:]

y_val = train_dataset_2[:10000]
partial_train_dataset_2 = train_dataset_2[10000:]

np.random.seed(123)

history_adam = model_adam.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.6990 - loss: 0.6165 - val_accuracy: 0.8547 - val_loss: 0.4033
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8923 - loss: 0.3308 - val_accuracy: 0.8830 - val_loss: 0.2999
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9293 - loss: 0.2122 - val_accuracy: 0.8894 - val_loss: 0.2767
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9536 - loss: 0.1535 - val_accuracy: 0.8874 - val_loss: 0.2822
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step - accuracy: 0.9689 - loss: 0.1139 - val_accuracy: 0.8834 - val_loss: 0.2976
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9788 - loss: 0.0852 - val_accuracy: 0.8827 - val_loss: 0.3198
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9873 - loss: 0.0631 - val_accuracy: 0.8809 - val_loss: 0.3473
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9931 - loss: 0.0464 - val_accuracy: 0.8784 - val_loss: 0.3795
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9960 - loss: 0.0324 - val_accuracy: 0.8761 - val_loss: 0.4141
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9981 - loss: 0.0230 - val_accuracy: 0.8740 - val_loss: 0.4464
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9992 - loss: 0.0162 - val_accuracy: 0.8742 - val_loss: 0.4801
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9998 - loss: 0.0110 - val_accuracy: 0.8728 - val_loss: 0.5082
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9994 - loss: 0.0089 - val_accuracy: 0.8722 - val_loss: 0.5347
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9999 - loss: 0.0062 - val_accuracy: 0.8718 - val_loss: 0.5572
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 1.0000 - loss: 0.0047 - val_accuracy: 0.8697 - val_loss: 0.5790
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9999 - loss: 0.0040 - val_accuracy: 0.8682 - val_loss: 0.5980
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 1.0000 - loss: 0.0030 - val_accuracy: 0.8687 - val_loss: 0.6150
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9999 - loss: 0.0026 - val_accuracy: 0.8688 - val_loss: 0.6318
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 1.0000 - loss: 0.0022 - val_accuracy: 0.8679 - val_loss: 0.6462
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 0.8689 - val_loss: 0.6592
```

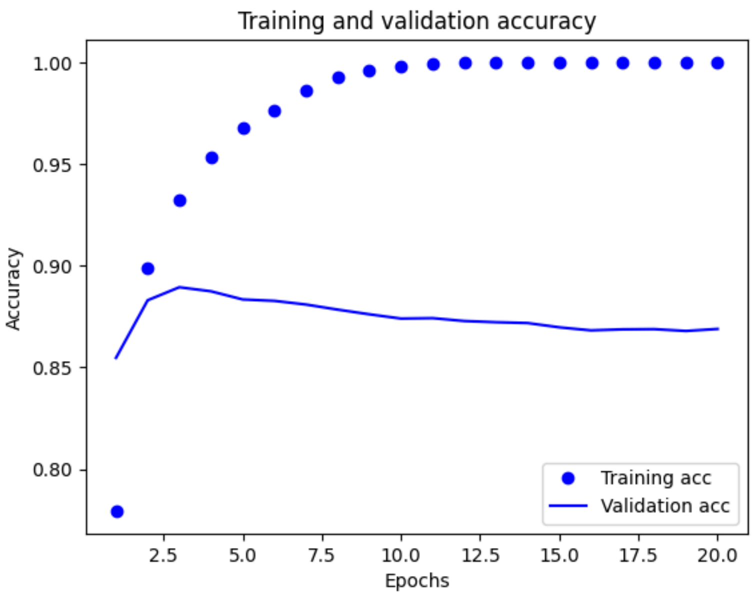
```
history_dict_adam = history_adam.history
history_dict_adam.keys()
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = history_dict_adam["loss"]
val_loss_values = history_dict_adam["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict_adam["accuracy"]
val_acc = history_dict_adam["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
model_adam.fit(train_dataset_1, train_dataset_2, epochs=4, batch_size=512)
binary_matrix_adam = model_adam.evaluate(test_dataset_1, test_data_2)
binary_matrix_adam
```

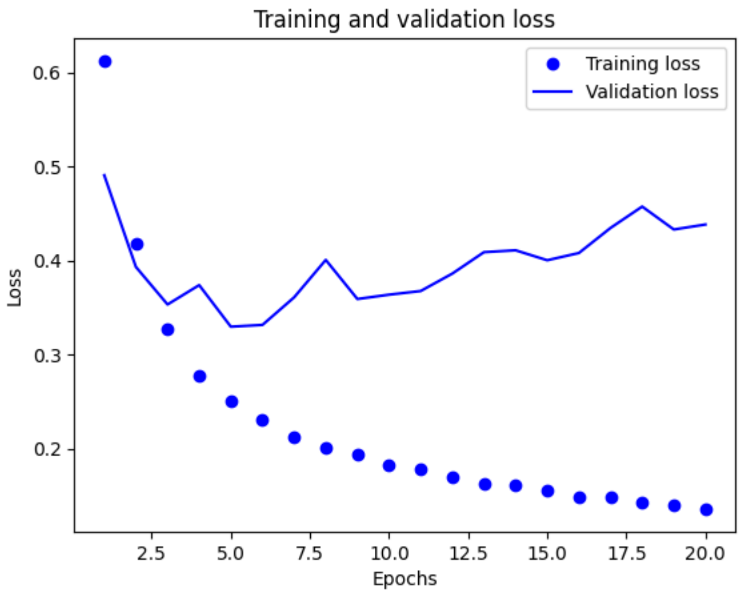
```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9450 - loss: 0.2408
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9721 - loss: 0.0972
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9852 - loss: 0.0568
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9930 - loss: 0.0329
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8578 - loss: 0.5264
[0.5296986103057861, 0.8575599789619446]
```

Regularization

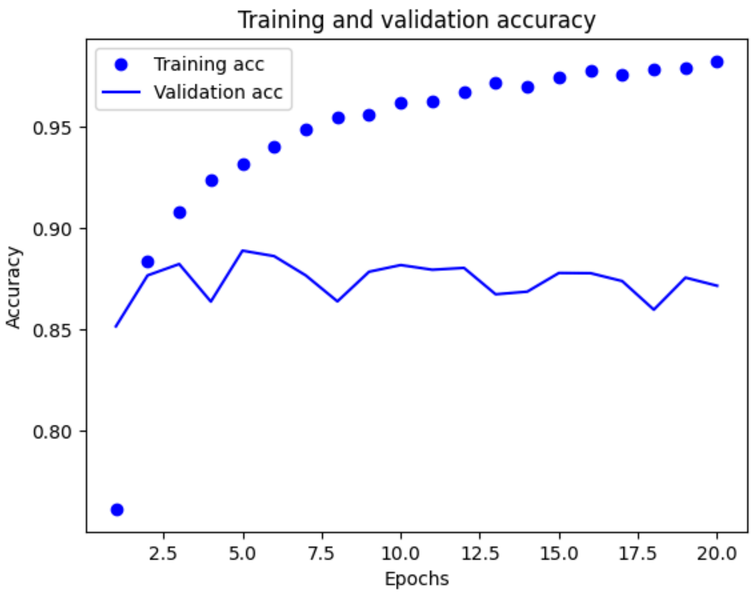
```
from tensorflow.keras import regularizers
np.random.seed(123)
model_regularization = keras.Sequential([
    layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
model_regularization.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
np.random.seed(123)
history_model_regularization = model_regularization.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
history_dict_regularization = history_model_regularization.history
history_dict_regularization.keys()
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.6741 - loss: 0.6740 - val_accuracy: 0.8516 - val_loss: 0.4908
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8805 - loss: 0.4418 - val_accuracy: 0.8767 - val_loss: 0.3932
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9066 - loss: 0.3357 - val_accuracy: 0.8823 - val_loss: 0.3532
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9259 - loss: 0.2783 - val_accuracy: 0.8638 - val_loss: 0.3739
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9323 - loss: 0.2501 - val_accuracy: 0.8889 - val_loss: 0.3296
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9483 - loss: 0.2222 - val_accuracy: 0.8862 - val_loss: 0.3315
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9520 - loss: 0.2072 - val_accuracy: 0.8767 - val_loss: 0.3608
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9597 - loss: 0.1939 - val_accuracy: 0.8639 - val_loss: 0.4007
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9566 - loss: 0.1945 - val_accuracy: 0.8785 - val_loss: 0.3591
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9675 - loss: 0.1728 - val_accuracy: 0.8818 - val_loss: 0.3637
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9677 - loss: 0.1702 - val_accuracy: 0.8795 - val_loss: 0.3676
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9722 - loss: 0.1629 - val_accuracy: 0.8804 - val_loss: 0.3861
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9751 - loss: 0.1562 - val_accuracy: 0.8674 - val_loss: 0.4089
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9735 - loss: 0.1550 - val_accuracy: 0.8687 - val_loss: 0.4109
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9800 - loss: 0.1460 - val_accuracy: 0.8779 - val_loss: 0.4003
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9829 - loss: 0.1390 - val_accuracy: 0.8778 - val_loss: 0.4080
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9816 - loss: 0.1386 - val_accuracy: 0.8739 - val_loss: 0.4347
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9820 - loss: 0.1367 - val_accuracy: 0.8598 - val_loss: 0.4574
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9822 - loss: 0.1359 - val_accuracy: 0.8756 - val_loss: 0.4330
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9871 - loss: 0.1267 - val_accuracy: 0.8716 - val_loss: 0.4383
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = history_dict_regularization["loss"]
val_loss_values = history_dict_regularization["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict_regularization["accuracy"]
val_acc = history_dict_regularization["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
model_regularization.fit(train_dataset_1, train_dataset_2, epochs=8, batch_size=512)
binary_matrix_regularization = model_regularization.evaluate(test_dataset_1, test_data_2)
binary_matrix_regularization
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9402 - loss: 0.2483
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9520 - loss: 0.2031
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9624 - loss: 0.1821
```

```
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9618 - loss: 0.1754
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9632 - loss: 0.1690
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9706 - loss: 0.1602
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9688 - loss: 0.1593
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9675 - loss: 0.1605
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8670 - loss: 0.4294
[0.42404553294181824, 0.8711199760437012]
```

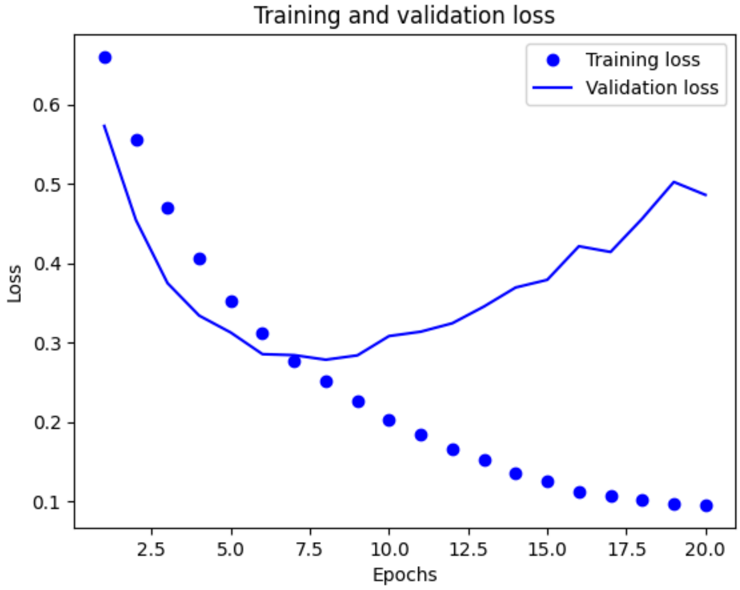
The model with regularization demonstrated steady improvement in training accuracy, reaching 96.75% with a loss of 0.1605 by Epoch 8. However, on the test set, it achieved an accuracy of 87.11% with a loss of 0.4240, indicating better generalization compared to previous models but still showing some overfitting. Further tuning, such as adjusting the regularization strength, may enhance performance.

Dropout

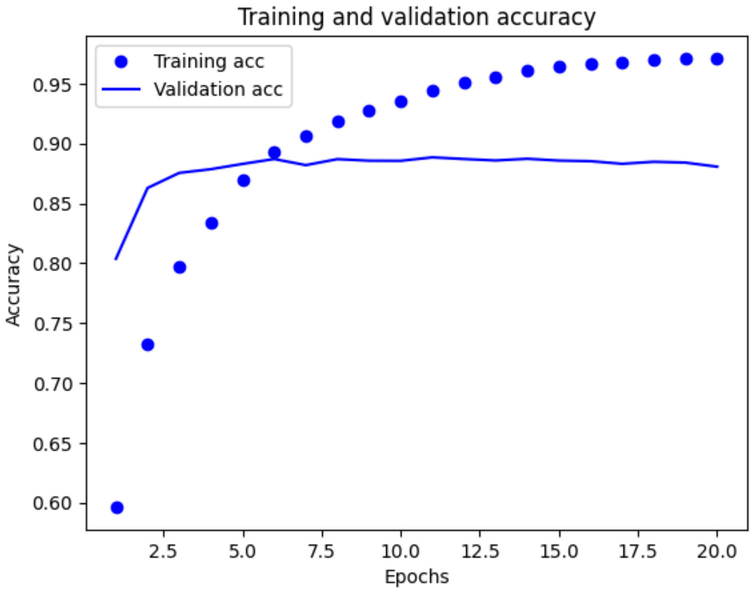
```
from tensorflow.keras import regularizers
np.random.seed(123)
model_Dropout = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model_Dropout.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
np.random.seed(123)
history_model_Dropout = model_Dropout.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
history_dict_Dropout = history_model_Dropout.history
history_dict_Dropout.keys()
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.5469 - loss: 0.6809 - val_accuracy: 0.8037 - val_loss: 0.5729
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.7137 - loss: 0.5756 - val_accuracy: 0.8629 - val_loss: 0.4544
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.7898 - loss: 0.4831 - val_accuracy: 0.8755 - val_loss: 0.3748
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8290 - loss: 0.4147 - val_accuracy: 0.8786 - val_loss: 0.3339
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8644 - loss: 0.3601 - val_accuracy: 0.8830 - val_loss: 0.3125
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8927 - loss: 0.3097 - val_accuracy: 0.8871 - val_loss: 0.2854
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9062 - loss: 0.2768 - val_accuracy: 0.8820 - val_loss: 0.2841
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9130 - loss: 0.2608 - val_accuracy: 0.8870 - val_loss: 0.2783
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9269 - loss: 0.2305 - val_accuracy: 0.8857 - val_loss: 0.2838
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9333 - loss: 0.2019 - val_accuracy: 0.8856 - val_loss: 0.3081
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9467 - loss: 0.1851 - val_accuracy: 0.8885 - val_loss: 0.3136
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9525 - loss: 0.1650 - val_accuracy: 0.8871 - val_loss: 0.3243
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9555 - loss: 0.1531 - val_accuracy: 0.8859 - val_loss: 0.3454
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9599 - loss: 0.1338 - val_accuracy: 0.8873 - val_loss: 0.3693
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9655 - loss: 0.1246 - val_accuracy: 0.8858 - val_loss: 0.3790
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9667 - loss: 0.1113 - val_accuracy: 0.8853 - val_loss: 0.4213
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9689 - loss: 0.1032 - val_accuracy: 0.8831 - val_loss: 0.4141
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9674 - loss: 0.1065 - val_accuracy: 0.8848 - val_loss: 0.4560
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9696 - loss: 0.0980 - val_accuracy: 0.8841 - val_loss: 0.5023
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9702 - loss: 0.0943 - val_accuracy: 0.8807 - val_loss: 0.4860
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = history_dict_Dropout["loss"]
val_loss_values = history_dict_Dropout["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = history_dict_Dropout["accuracy"]
val_acc = history_dict_Dropout["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
model_Dropout.fit(train_dataset_1, train_dataset_2, epochs=8, batch_size=512)
binary_matrix_Dropout = model_Dropout.evaluate(test_dataset_1, test_data_2)
binary_matrix_Dropout
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9265 - loss: 0.2493
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9341 - loss: 0.2052
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9438 - loss: 0.1895
```

```
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9462 - loss: 0.1668
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9501 - loss: 0.1495
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - accuracy: 0.9517 - loss: 0.1475
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9506 - loss: 0.1412
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9554 - loss: 0.1299
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8744 - loss: 0.4879
[0.4821183681488037, 0.8751999735832214]
```

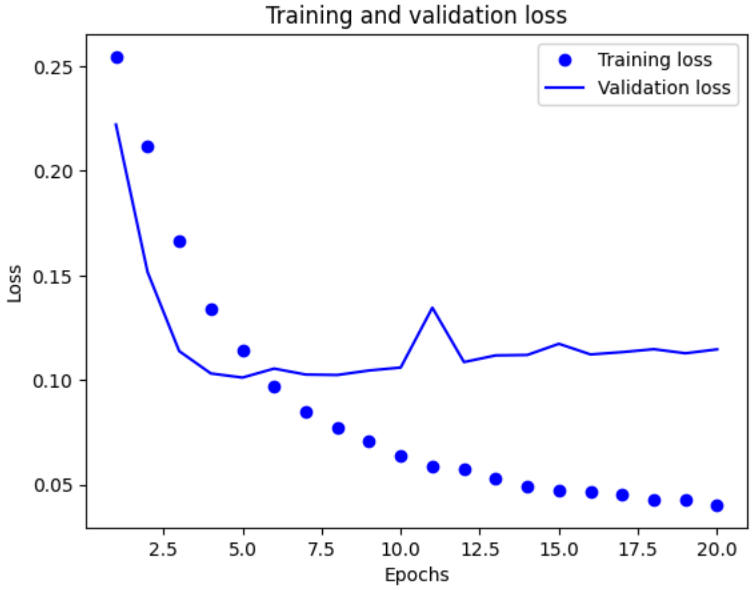
The model showed consistent improvement during training, reaching an accuracy of 95.54% with a loss of 0.1299 by Epoch 8. On the test set, it achieved 87.52% accuracy with a loss of 0.4821, indicating decent generalization but still exhibiting signs of overfitting. Further fine-tuning, such as adjusting regularization strength or dropout, could help enhance performance.

Training model with hyper tuned parameters

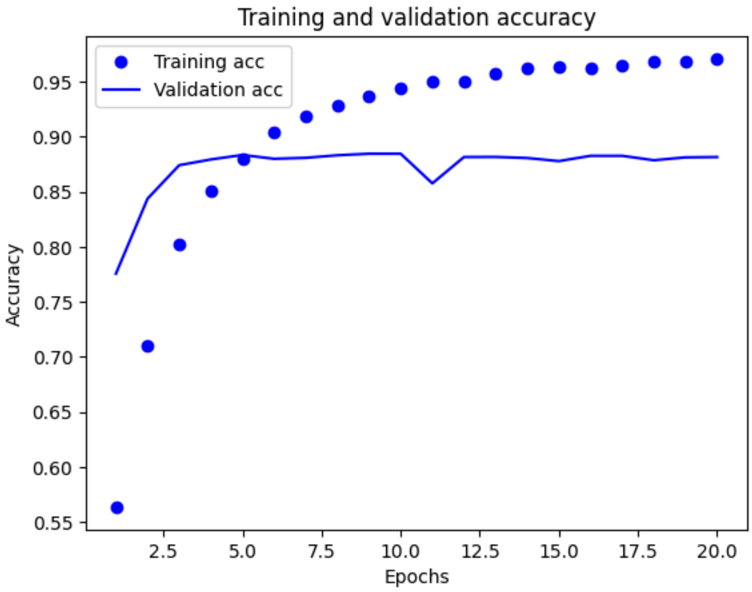
```
from tensorflow.keras import regularizers
np.random.seed(123)
hyper_model = keras.Sequential([
    layers.Dense(32, activation="relu",kernel_regularizer=regularizers.l2(0.0001)),
    layers.Dropout(0.5),
    layers.Dense(32, activation="relu",kernel_regularizer=regularizers.l2(0.0001)),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.0001)),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
hyper_model.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
np.random.seed(123)
history_model_Hyper = hyper_model.fit(partial_train_dataset_1,
                    partial_train_dataset_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_validation, y_val))
hyper_model_history_dict = history_model_Hyper.history
hyper_model_history_dict.keys()
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 51ms/step - accuracy: 0.5316 - loss: 0.2594 - val_accuracy: 0.7755 - val_loss: 0.2221
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.6811 - loss: 0.2225 - val_accuracy: 0.8436 - val_loss: 0.1516
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.7873 - loss: 0.1733 - val_accuracy: 0.8740 - val_loss: 0.1139
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.8455 - loss: 0.1373 - val_accuracy: 0.8792 - val_loss: 0.1032
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8780 - loss: 0.1151 - val_accuracy: 0.8833 - val_loss: 0.1012
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9035 - loss: 0.0998 - val_accuracy: 0.8798 - val_loss: 0.1055
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9153 - loss: 0.0870 - val_accuracy: 0.8807 - val_loss: 0.1027
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9254 - loss: 0.0785 - val_accuracy: 0.8830 - val_loss: 0.1025
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9342 - loss: 0.0722 - val_accuracy: 0.8844 - val_loss: 0.1046
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9455 - loss: 0.0633 - val_accuracy: 0.8844 - val_loss: 0.1060
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9486 - loss: 0.0599 - val_accuracy: 0.8575 - val_loss: 0.1346
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9452 - loss: 0.0612 - val_accuracy: 0.8814 - val_loss: 0.1086
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9550 - loss: 0.0541 - val_accuracy: 0.8815 - val_loss: 0.1118
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9651 - loss: 0.0460 - val_accuracy: 0.8805 - val_loss: 0.1120
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9637 - loss: 0.0467 - val_accuracy: 0.8777 - val_loss: 0.1174
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9610 - loss: 0.0470 - val_accuracy: 0.8825 - val_loss: 0.1123
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9626 - loss: 0.0465 - val_accuracy: 0.8825 - val_loss: 0.1134
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9686 - loss: 0.0425 - val_accuracy: 0.8785 - val_loss: 0.1148
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9701 - loss: 0.0416 - val_accuracy: 0.8811 - val_loss: 0.1128
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9703 - loss: 0.0399 - val_accuracy: 0.8814 - val_loss: 0.1147
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
loss_values = hyper_model_history_dict["loss"]
val_loss_values = hyper_model_history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc = hyper_model_history_dict["accuracy"]
val_acc = hyper_model_history_dict["val_accuracy"]
plt.plot(epochs, acc, "bo", label="Training acc")
plt.plot(epochs, val_acc, "b", label="Validation acc")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
hyper_model.fit(train_dataset_1, train_dataset_2, epochs=8, batch_size=512)
binary_predictions = hyper_model.evaluate(test_dataset_1, test_data_2)
binary_predictions
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9282 - loss: 0.0737
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9342 - loss: 0.0683
Epoch 3/8
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9429 - loss: 0.0621
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9459 - loss: 0.0592
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9518 - loss: 0.0540
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9521 - loss: 0.0538
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9517 - loss: 0.0534
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9535 - loss: 0.0521
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8770 - loss: 0.1158
[0.11340228468179703, 0.8803600072860718]
```

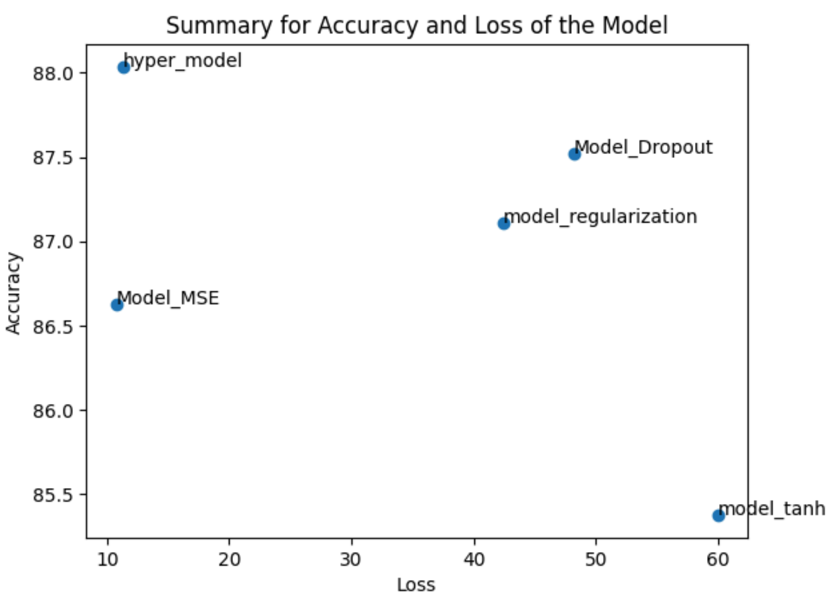
Summary

```
All_Models_Loss= np.array([binary_matrix_Dropout[0],binary_predictions[0],binary_mse[0],binary_matrix_regularization[0],bina
All_Models_Loss
All_Models_Accuracy= np.array([binary_matrix_Dropout[1],binary_predictions[1],binary_mse[1],binary_matrix_regularization[1],
All_Models_Accuracy
model_labels=['Model_Dropout','hyper_model','Model_MSE','model_regularization','model_tanh']
plt.clf()
```

```
<Figure size 640x480 with 0 Axes>
```

Compilation

```
fig, ax = plt.subplots()
ax.scatter(All_Models_Loss,All_Models_Accuracy)
for i, txt in enumerate(model_labels):
    ax.annotate(txt, (All_Models_Loss[i],All_Models_Accuracy[i] ))
plt.title("Summary for Accuracy and Loss of the Model")
plt.ylabel("Accuracy")
plt.xlabel("Loss")

plt.show()
```



Summary

Analysis of Model Performance: Accuracy vs. Loss The graph presents a comparative evaluation of different model variations based on accuracy and loss metrics. Each point represents a model with specific optimization techniques, illustrating their effectiveness in balancing accuracy and generalization.

Key Insights:

Hyperparameter-Tuned Model (hyper_model)

Achieves the highest accuracy (~88%) with the lowest loss, making it the best-performing model. This suggests that optimized hyperparameters significantly enhance both accuracy and generalization.

Model with Mean Squared Error (Model_MSE)

Demonstrates good accuracy (~86.5%) and the lowest loss, highlighting the effectiveness of MSE as a loss function in stabilizing training and reducing overfitting.

Dropout Regularization (Model_Dropout)

Achieves slightly higher accuracy (~87.5%) than other regularized models but with a moderate loss. Indicates that dropout helps improve generalization while maintaining strong performance.

L2 Regularization (model_regularization)

Attains an accuracy of ~87% with slightly higher loss compared to the dropout model. Suggests that L2 regularization aids in controlling overfitting but does not outperform hyperparameter tuning or dropout.

Tanh Activation Model (model_tanh)

Yields the lowest accuracy (85.5%) and the highest loss (~60), making it the least effective model. Implies that ReLU-based architectures were more efficient than tanh for this dataset.

Conclusion:

The hyperparameter-tuned model outperforms all others, achieving the highest accuracy with the lowest loss. Dropout and L2 regularization improve generalization, though dropout offers slightly better results. MSE as a loss function stabilizes training, reducing loss while maintaining competitive accuracy. Tanh activation underperforms, reinforcing the superiority of ReLU-based architectures for this task. For further optimization, combining hyperparameter tuning with dropout or L2 regularization could enhance performance while mitigating overfitting.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.