In [1]:
```
from numpy.random import seed
seed(123)
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
num_words=10000)
```

In [2]:
```
train_data
```

```
Out[2]: 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, 17
        2, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 5
        0, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 53
        0, 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, 553
        5, 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, 9
        52, 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, 138
        2, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 7
        8, 285, 16, 145, 95]),
               list([1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 5974, 54, 61, 369, 13, 7
        1, 149, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829, 296, 4, 86, 32
        0, 35, 534, 19, 263, 4821, 1301, 4, 1873, 33, 89, 78, 12, 66, 16, 4, 360, 7, 4, 5
        8, 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, 1
        2, 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, 400
        7, 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, 81
        23, 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, 15
        74, 7, 496, 4, 139, 929, 2901, 2, 7750, 5, 4241, 18, 4, 8497, 2, 250, 11, 1818, 75
        61, 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, 48
        4, 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, 219
        8, 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, 5
        2, 58, 2, 92, 401, 728, 12, 39, 14, 251, 8, 15, 251, 5, 2, 12, 38, 84, 80, 124, 1
        2, 9, 23]),
               list([1, 17, 6, 194, 337, 7, 4, 204, 22, 45, 254, 8, 106, 14, 123, 4, 2, 27
        0, 2, 5, 2, 2, 732, 2098, 101, 405, 39, 14, 1034, 4, 1310, 9, 115, 50, 305, 12, 4
        7, 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, 1
0, 10, 12, 764, 40, 4, 248, 20, 12, 16, 5, 174, 1791, 72, 7, 51, 6, 1739, 22, 4, 2
04, 131, 9])],
      dtype=object)
```

In [3]:
```
train_labels[0]
```

Out[3]:
```
1
```

In [4]:
```
 len(train_labels)
```

Out[4]:
```
25000
```

In [5]:
```
test_data
```

```
Out[5]: array([list([1, 591, 202, 14, 31, 6, 717, 10, 10, 2, 2, 5, 4, 360, 7, 4, 177, 576
0, 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, 94
3, 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, 5
0, 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, 207
8, 7, 5293, 861, 2, 5, 4182, 30, 3127, 2, 56, 4, 841, 5, 990, 692, 8, 4, 1669, 39
8, 229, 10, 10, 13, 2822, 670, 5304, 14, 9, 31, 7, 27, 111, 108, 15, 2033, 19, 783
6, 1429, 875, 551, 14, 22, 9, 1193, 21, 45, 4829, 5, 45, 252, 8, 2, 6, 565, 921, 3
639, 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, 99
7, 1638, 8, 35, 2076, 9019, 11, 22, 231, 54, 29, 1706, 29, 100, 2, 2425, 34, 2, 87
38, 2, 5, 2, 98, 31, 2122, 33, 6, 58, 14, 3808, 1638, 8, 4, 365, 7, 2789, 3761, 35
6, 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, 47
6, 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, 71
49, 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, 1
68, 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, 609
3, 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, 1
9, 6, 2, 7, 2, 1810, 2, 4, 4770, 3183, 930, 8, 508, 90, 4, 1317, 8, 4, 2, 17, 2, 3
965, 1853, 4, 1494, 8, 4468, 189, 4, 2, 6287, 5774, 4, 4770, 5, 95, 271, 23, 6, 77
42, 6063, 2, 5437, 33, 1526, 6, 425, 3155, 2, 4535, 1636, 7, 4, 4669, 2, 469, 4, 4
552, 54, 4, 150, 5664, 2, 280, 53, 2, 2, 18, 339, 29, 1978, 27, 7885, 5, 2, 68, 18
30, 19, 6571, 2, 4, 1515, 7, 263, 65, 2132, 34, 6, 5680, 7489, 43, 159, 29, 9, 470
6, 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, 328
6, 325, 12, 62, 30, 776, 8, 67, 14, 17, 6, 2, 44, 148, 687, 2, 203, 42, 203, 24, 2
8, 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, 2
1, 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, 11
```

```
9, 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, 3
33, 20, 4, 985, 200, 5, 2, 5, 9, 1861, 8, 79, 357, 4, 20, 47, 220, 57, 206, 139, 1
1, 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, 1
5, 47, 6, 3482, 4067, 8, 114, 5, 33, 222, 31, 55, 184, 704, 5586, 2, 19, 346, 315
3, 5, 6, 364, 350, 4, 184, 5586, 9, 133, 1810, 11, 5417, 2, 21, 4, 7298, 2, 570, 5
0, 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, 2
740, 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, 38
2, 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)
```

In [6]:
```
test_labels[0]
```

Out[6]:
```
0
```

In [7]:
```
 max([max(sequence) for sequence in test_data])
```

Out[7]:
```
9999
```

text review

In [8]:
```
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = " ".join([reverse_word_index.get(i - 3, "?") for i in train_data[0
```

In [ ]:

In [9]:
```
decoded_review
```

Out[9]:
```
"? this film was just brilliant casting location scenery story direction everyon
e's really suited the part they played and you could just imagine being there robe
rt ? is an amazing actor and now the same being director ? father came from the sa
me scottish island as myself so i loved the fact there was a real connection with
this film the witty remarks throughout the film were great it was just brilliant s
o much that i bought the film as soon as it was released for ? and would recommend
it to everyone to watch and the fly fishing was amazing really cried at the end it
was so sad and you know 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 that played the ? of norman
and paul they were just brilliant children are often left out of the ? list i thin
k because the stars that play them all grown up are such a big profile for the who
le film but these children are amazing and should be praised for what they have do
ne don't you think the whole story was so lovely because it was true and was someo
ne's life after all that was shared with us all"
```

preparation of Data

In [10]:
```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
 results = np.zeros((len(sequences), dimension))
 for i, sequence in enumerate(sequences):
  for j in sequence:
   results[i, j] = 1.
 return results
```

Vectorization of Data

In [11]:
```
a_train = vectorize_sequences(train_data)
a_test = vectorize_sequences(test_data)
```

In [12]:
```
a_train[0]
```

Out[12]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

In [13]:
```
a_test[0]
```

Out[13]:
```
array([0., 1., 1., ..., 0., 0., 0.])
```

Labeling the vectorization

In [14]:
```
b_train = np.asarray(train_labels).astype("float32")
b_test = np.asarray(test_labels).astype("float32")
```

Developing model using relu and compiling it

In [15]:
```
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")
])
```

In [16]:
```
 model.compile(optimizer="rmsprop",
loss="binary_crossentropy",
metrics=["accuracy"])
```

In [17]:
```
 seed(123)
a_val = a_train[:10000]
partial_a_train =a_train[10000:]
b_val = b_train[:10000]
partial_b_train = b_train[10000:]
```

In [18]:
```
seed(123)
history = model.fit(partial_a_train,
partial_b_train,
epochs=20,
```

```
batch_size=512,
validation_data=(a_val, b_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 8s 181ms/step - accuracy: 0.6805 - loss: 0.6150 - val_acc
uracy: 0.8584 - val_loss: 0.4198
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.8882 - loss: 0.3625 - val_accu
racy: 0.8684 - val_loss: 0.3436
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9190 - loss: 0.2647 - val_accu
racy: 0.8867 - val_loss: 0.2888
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9366 - loss: 0.2095 - val_accu
racy: 0.8896 - val_loss: 0.2775
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9442 - loss: 0.1726 - val_accu
racy: 0.8818 - val_loss: 0.2933
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9524 - loss: 0.1485 - val_accu
racy: 0.8835 - val_loss: 0.2842
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9669 - loss: 0.1204 - val_accu
racy: 0.8850 - val_loss: 0.2967
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9670 - loss: 0.1084 - val_accu
racy: 0.8838 - val_loss: 0.3015
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9757 - loss: 0.0879 - val_accu
racy: 0.8773 - val_loss: 0.3269
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9813 - loss: 0.0785 - val_accu
racy: 0.8822 - val_loss: 0.3315
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9843 - loss: 0.0673 - val_accu
racy: 0.8818 - val_loss: 0.3494
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9855 - loss: 0.0582 - val_accu
racy: 0.8768 - val_loss: 0.3848
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9904 - loss: 0.0476 - val_accu
racy: 0.8760 - val_loss: 0.3916
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9932 - loss: 0.0388 - val_accu
racy: 0.8690 - val_loss: 0.4529
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9936 - loss: 0.0366 - val_accu
racy: 0.8739 - val_loss: 0.4341
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9962 - loss: 0.0285 - val_accu
racy: 0.8733 - val_loss: 0.4633
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9976 - loss: 0.0234 - val_accu
racy: 0.8734 - val_loss: 0.4724
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9985 - loss: 0.0198 - val_accu
racy: 0.8665 - val_loss: 0.5318
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9986 - loss: 0.0171 - val_accu
```

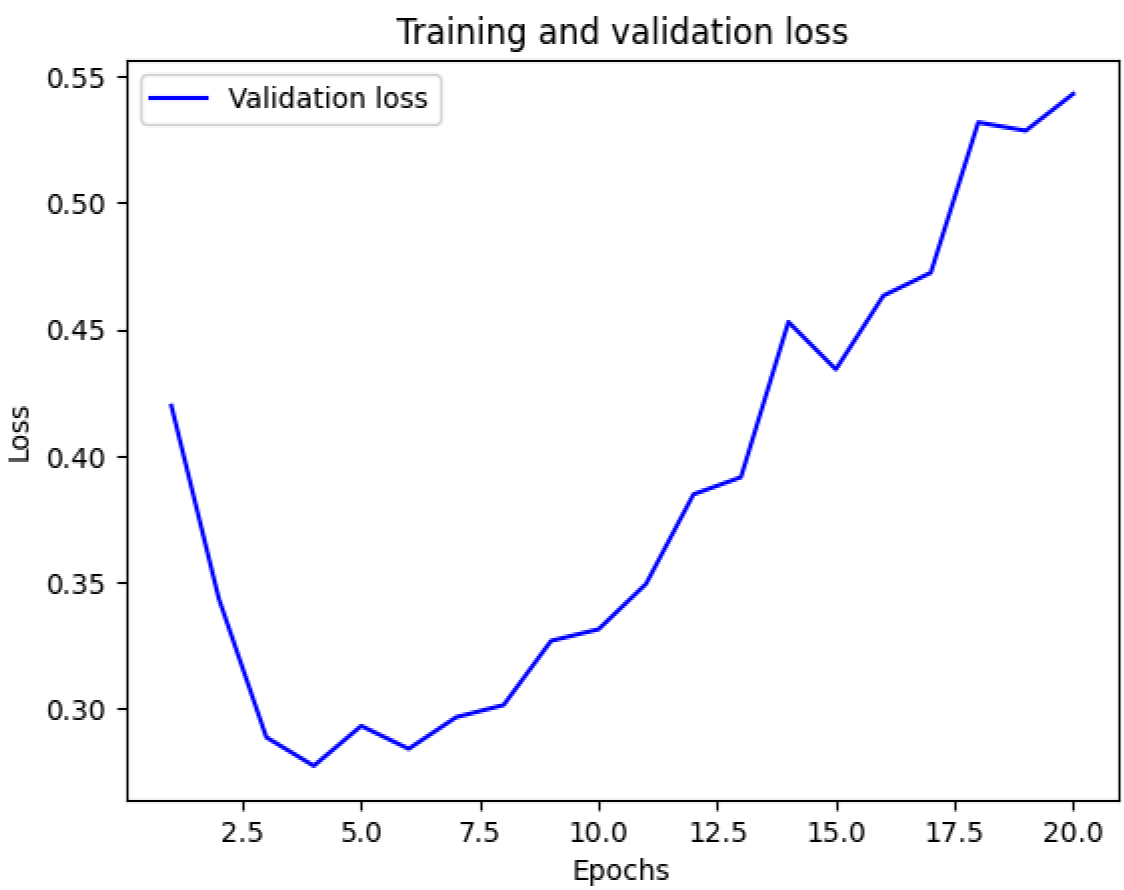
```
racy: 0.8693 - val_loss: 0.5284
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9989 - loss: 0.0140 - val_accu
racy: 0.8732 - val_loss: 0.5430
```

In [19]:
```
history_dict = history.history
history_dict.keys()
```

Out[19]: `dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])`

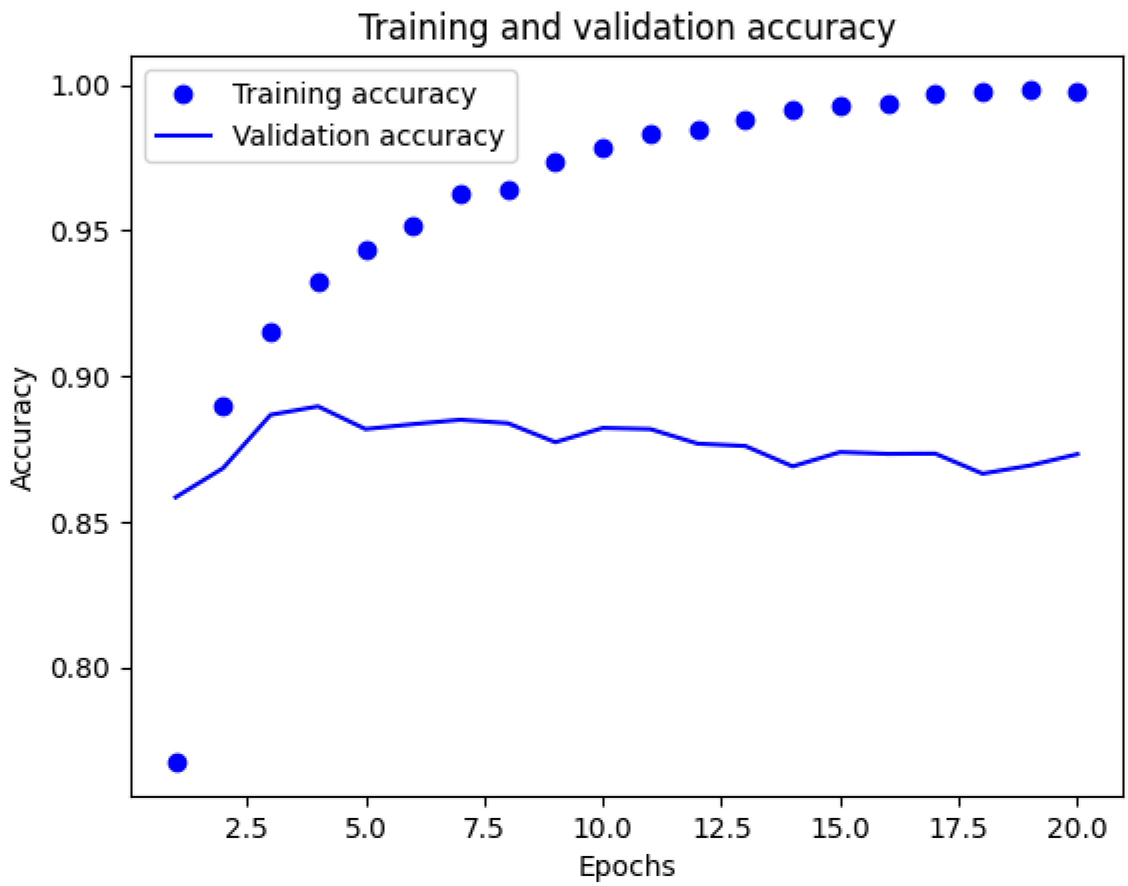
plotting the training and validation loss

In [20]:
```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
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()
```



In [21]:
```
plt.clf()
acc = history_dict["accuracy"]
val_acc = history_dict["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 two graphs indicate that overfitting the training data reduces the model's ability to predict new data after a certain epoch. To enhance the model's performance, it may be essential to further refine the analysis, such as adjusting hyperparameters or employing techniques like regularization.

Retraining the model

In [22]:
```
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(a_train, b_train, epochs=4, batch_size=512)
results = model.evaluate(a_test, b_test)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.7145 - loss: 0.5538
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9083 - loss: 0.2726
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9245 - loss: 0.2104
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9380 - loss: 0.1772
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8825 - loss: 0.2916
```

In [23]:
```
results
```

Out[23]:
```
[0.29043278098106384, 0.883840024471283]
```

-

from test dataset model achieved accuracy of 88.72% and loss is 0.28

In [24]:
```
model.predict(a_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
```

Out[24]:
```
array([[0.14741957],
       [0.99947673],
       [0.7795537 ],
       ...,
       [0.08244462],
       [0.08229432],
       [0.49783552]], dtype=float32)
```

Building a neural network with 1 hidden layer

In [25]:
```
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_val = a_train[:10000]
partial_a_train = a_train[10000:]
a_val = b_train[:10000]
partial_b_train = b_train[10000:]
history1 = model1.fit(partial_a_train,
                    partial_b_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, a_val))
```

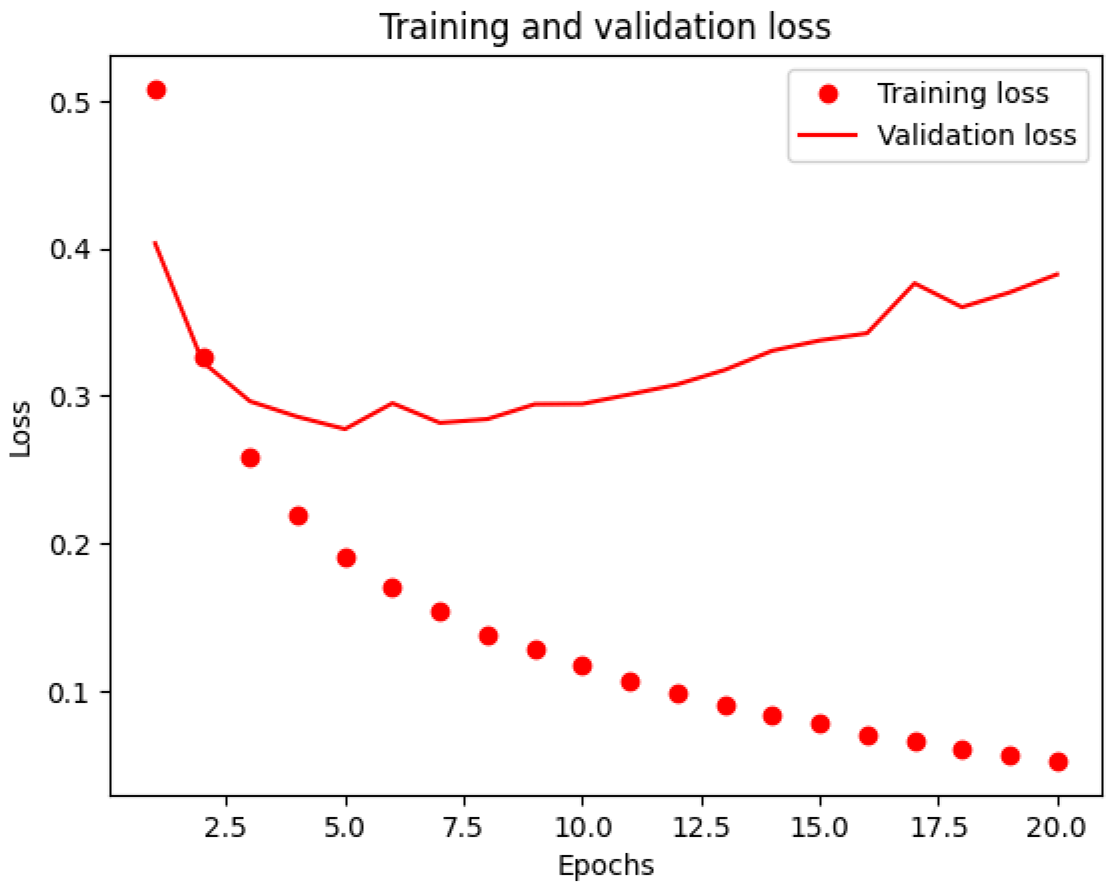
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 188ms/step - accuracy: 0.6989 - loss: 0.5853 - val_acc
uracy: 0.8553 - val_loss: 0.4033
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.8928 - loss: 0.3447 - val_accu
racy: 0.8822 - val_loss: 0.3229
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9200 - loss: 0.2610 - val_accu
racy: 0.8863 - val_loss: 0.2962
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9309 - loss: 0.2213 - val_accu
racy: 0.8886 - val_loss: 0.2857
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9422 - loss: 0.1889 - val_accu
racy: 0.8890 - val_loss: 0.2774
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9490 - loss: 0.1673 - val_accu
racy: 0.8812 - val_loss: 0.2949
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9527 - loss: 0.1532 - val_accu
racy: 0.8853 - val_loss: 0.2816
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9614 - loss: 0.1361 - val_accu
racy: 0.8842 - val_loss: 0.2842
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9645 - loss: 0.1260 - val_accu
racy: 0.8799 - val_loss: 0.2943
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9712 - loss: 0.1136 - val_accu
racy: 0.8850 - val_loss: 0.2945
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9733 - loss: 0.1064 - val_accu
racy: 0.8847 - val_loss: 0.3009
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9769 - loss: 0.0967 - val_accu
racy: 0.8824 - val_loss: 0.3077
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9799 - loss: 0.0861 - val_accu
racy: 0.8790 - val_loss: 0.3175
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9821 - loss: 0.0809 - val_accu
racy: 0.8759 - val_loss: 0.3305
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9832 - loss: 0.0755 - val_accu
racy: 0.8760 - val_loss: 0.3375
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9869 - loss: 0.0667 - val_accu
racy: 0.8777 - val_loss: 0.3424
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9876 - loss: 0.0631 - val_accu
racy: 0.8678 - val_loss: 0.3762
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9897 - loss: 0.0594 - val_accu
racy: 0.8778 - val_loss: 0.3601
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9902 - loss: 0.0537 - val_accu
```

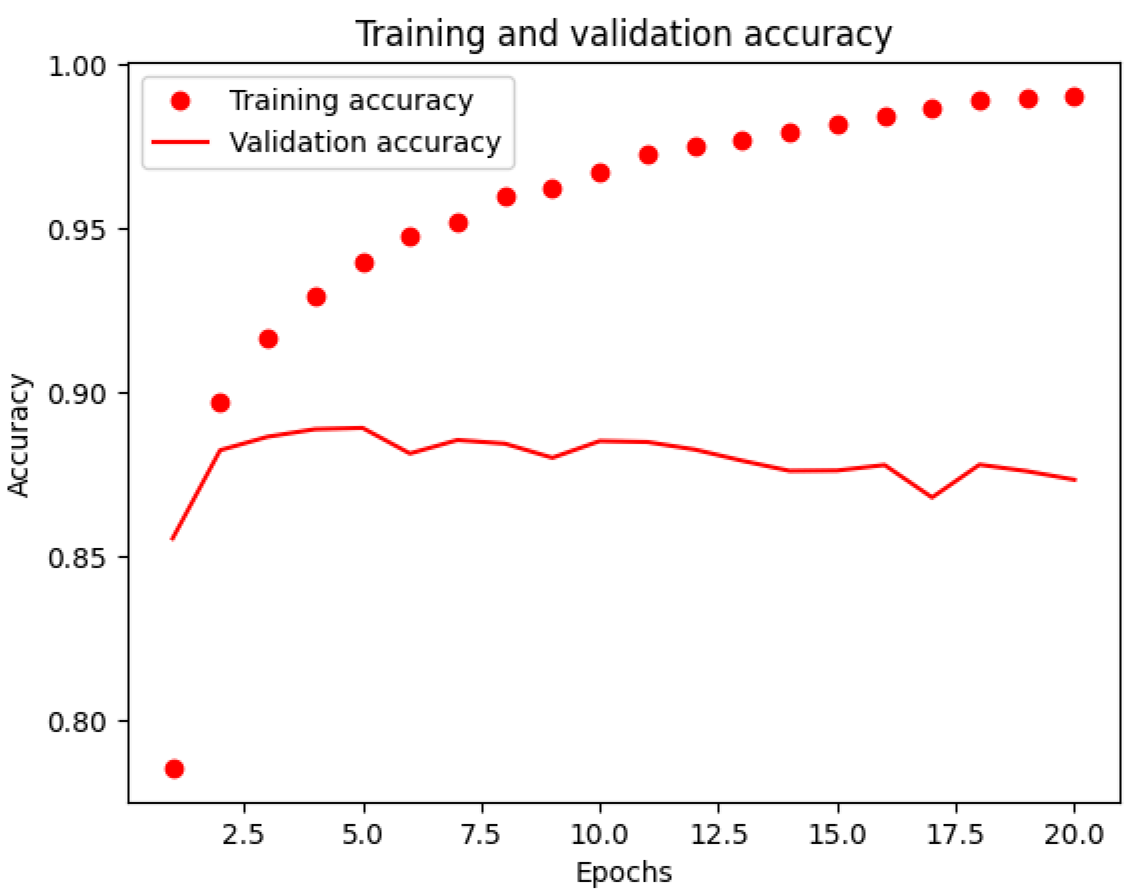
```
racy: 0.8758 - val_loss: 0.3700
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9923 - loss: 0.0489 - val_accu
racy: 0.8732 - val_loss: 0.3823
```

In [26]:
```
history_dict = history1.history
history_dict.keys()
```

Out[26]: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])

In [27]:
```
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()
```


Training and validation loss
Training loss
Validation loss
Loss
Epochs
0.5
0.4
0.3
0.2
0.1
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Training and validation accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs
1.00
0.95
0.90
0.85
0.80
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

In [28]:
```
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(a_train, b_train, epochs=5, batch_size=512)
results1 = model1.evaluate(a_test, b_test)
```

```
Epoch 1/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.7459 - loss: 0.5455
Epoch 2/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8990 - loss: 0.3051
Epoch 3/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9212 - loss: 0.2394
Epoch 4/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9290 - loss: 0.2090
Epoch 5/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9374 - loss: 0.1847
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8863 - loss: 0.2811
```

In [29]:
```
results1
```

Out[29]:
```
[0.27939900755882263, 0.888159990310669]
```

the test resulted with loss of 0.28 and accuracy is 88.7%

In [30]:
```
model1.predict(a_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
```

Out[30]:
```
array([[0.26877818],
       [0.9998534 ],
       [0.840057  ],
       ...,
       [0.1523046 ],
       [0.1033494 ],
       [0.60843086]], dtype=float32)
```

neural network with 3 hidden layers

In [31]:
```
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"])
a_val = a_train[:10000]
partial_a_train = a_train[10000:]
b_val = b_train[:10000]
```

```
partial_b_train = b_train[10000:]
history3 = model_3.fit(partial_a_train,
partial_b_train,
                        epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
```

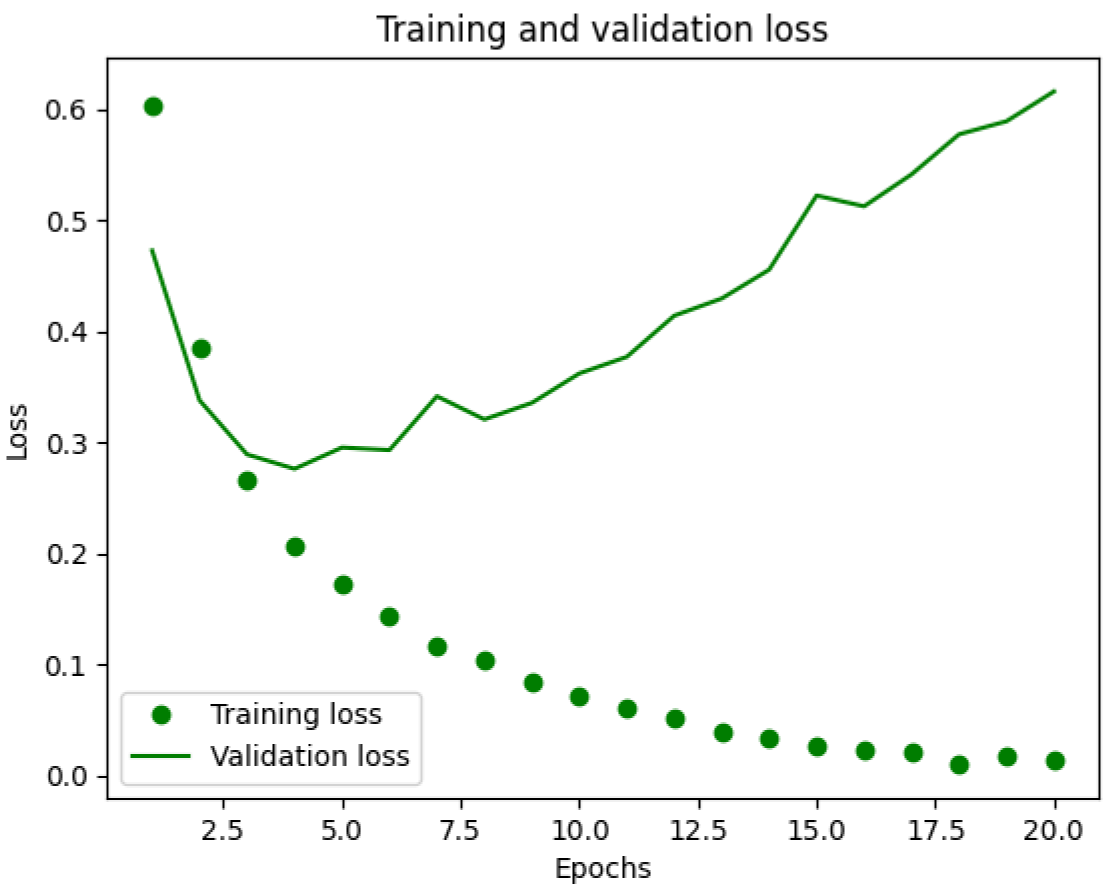
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 8s 203ms/step - accuracy: 0.6363 - loss: 0.6548 - val_acc
uracy: 0.8595 - val_loss: 0.4728
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.8778 - loss: 0.4166 - val_accu
racy: 0.8817 - val_loss: 0.3379
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9150 - loss: 0.2764 - val_accu
racy: 0.8923 - val_loss: 0.2892
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9324 - loss: 0.2093 - val_accu
racy: 0.8904 - val_loss: 0.2762
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9402 - loss: 0.1725 - val_accu
racy: 0.8836 - val_loss: 0.2955
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9551 - loss: 0.1403 - val_accu
racy: 0.8843 - val_loss: 0.2932
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9677 - loss: 0.1097 - val_accu
racy: 0.8769 - val_loss: 0.3416
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9691 - loss: 0.0994 - val_accu
racy: 0.8853 - val_loss: 0.3208
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9790 - loss: 0.0807 - val_accu
racy: 0.8829 - val_loss: 0.3356
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9837 - loss: 0.0676 - val_accu
racy: 0.8818 - val_loss: 0.3621
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9860 - loss: 0.0552 - val_accu
racy: 0.8781 - val_loss: 0.3771
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9891 - loss: 0.0490 - val_accu
racy: 0.8794 - val_loss: 0.4141
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9927 - loss: 0.0374 - val_accu
racy: 0.8777 - val_loss: 0.4296
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9939 - loss: 0.0294 - val_accu
racy: 0.8736 - val_loss: 0.4555
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9967 - loss: 0.0226 - val_accu
racy: 0.8700 - val_loss: 0.5223
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9972 - loss: 0.0202 - val_accu
racy: 0.8750 - val_loss: 0.5126
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9982 - loss: 0.0160 - val_accu
racy: 0.8737 - val_loss: 0.5413
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9994 - loss: 0.0105 - val_accu
racy: 0.8663 - val_loss: 0.5774
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9936 - loss: 0.0227 - val_accu
```

```
racy: 0.8716 - val_loss: 0.5891
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9993 - loss: 0.0068 - val_accu
racy: 0.8733 - val_loss: 0.6160
```

In [32]:
```
history_dict3 = history3.history
history_dict3.keys()
```

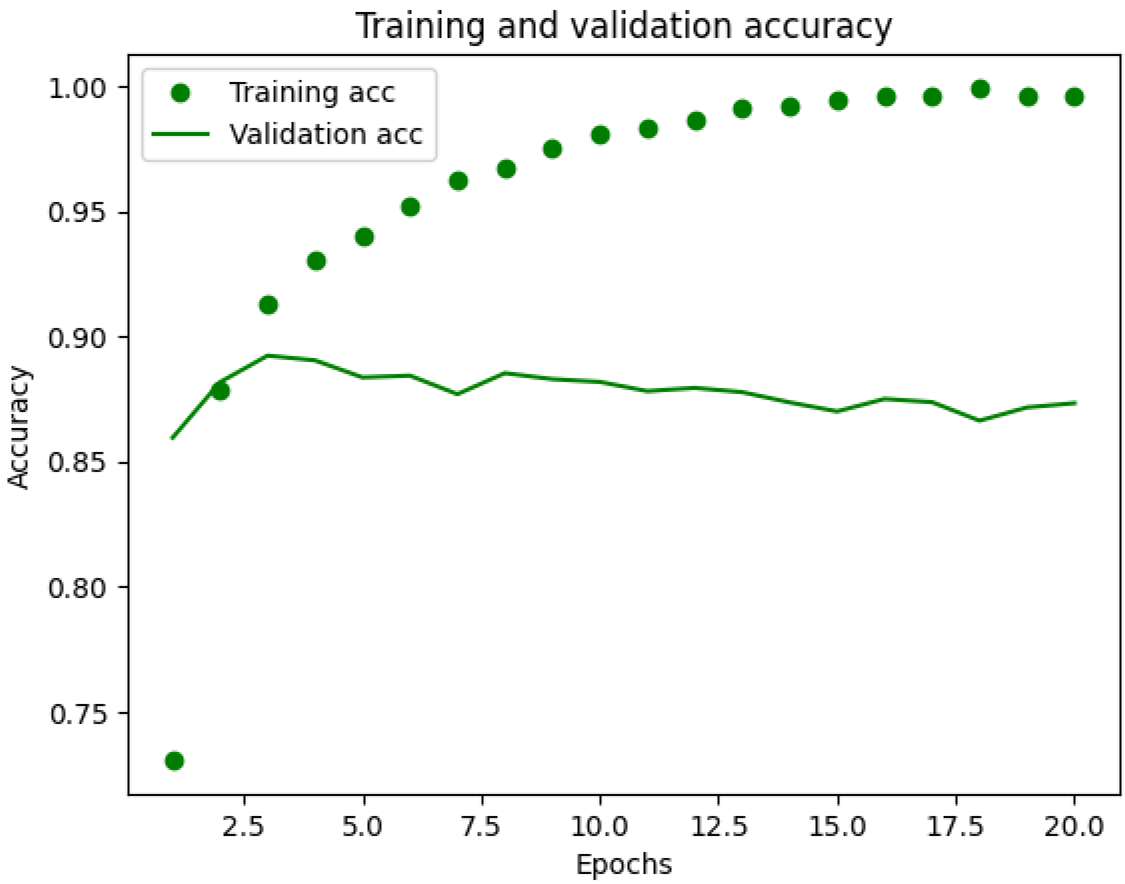
Out[32]: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])

In [33]:
```
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()
```



In [34]:
```
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()
```



In [35]:
```
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(a_train, b_train, epochs=3, batch_size=512)
results_3 = model_3.evaluate(a_test, b_test)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.7304 - loss: 0.5616
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9023 - loss: 0.2734
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9245 - loss: 0.2041
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8788 - loss: 0.2954
```

In [36]:
```
results_3
```

Out[36]:
```
[0.2949553430080414, 0.8807200193405151]
```

In [37]:
```
model_3.predict(a_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
```

Out[37]:
```
array([[0.19185777],
       [0.9997913 ],
       [0.62618005],
       ...,
       [0.10093132],
       [0.07036752],
       [0.61114585]], dtype=float32)
```

Neural Network with 32 units.

In [38]:
```
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
a_val = a_train[:10000]
partial_a_train = a_train[10000:] # Corrected line: Selecting data from a_train ins
b_val = b_train[:10000]
partial_b_train = b_train[10000:]
np.random.seed(123)
history32 = model_32.fit(partial_a_train,
partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
```

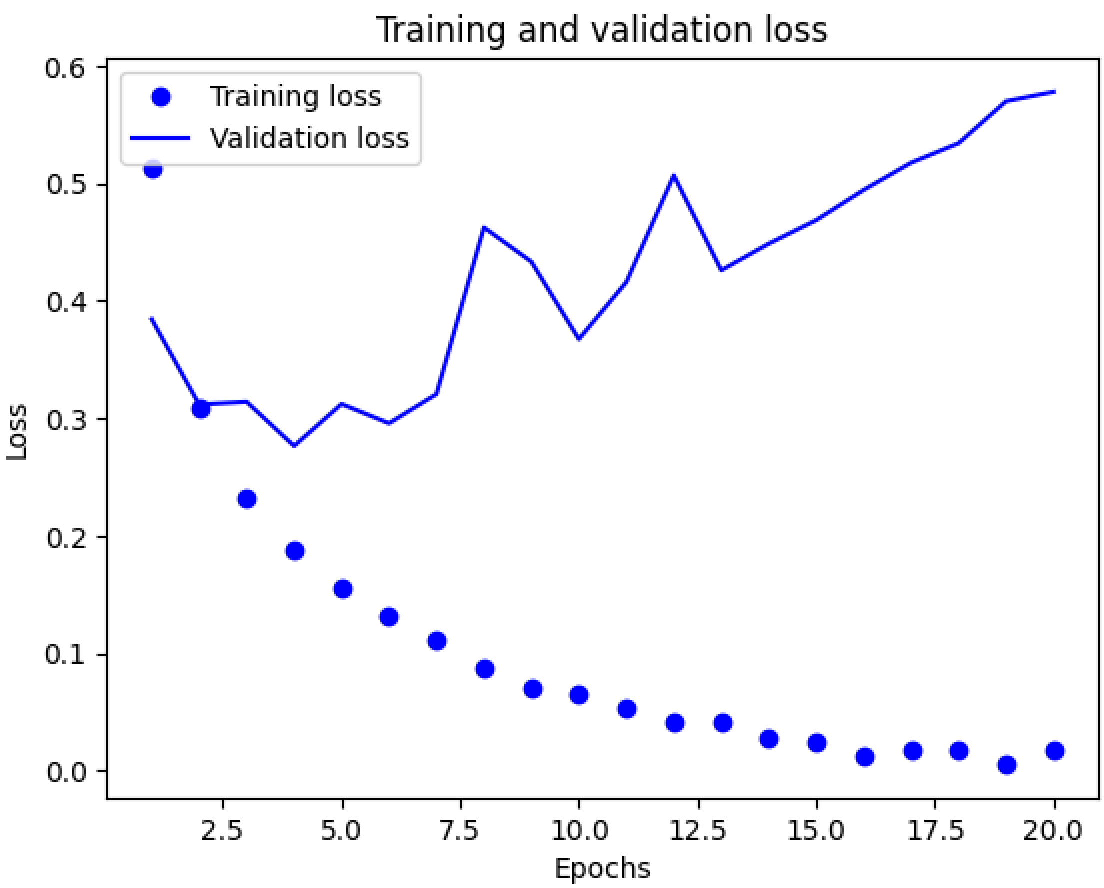
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 9s 227ms/step - accuracy: 0.6959 - loss: 0.5978 - val_acc
uracy: 0.8542 - val_loss: 0.3845
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - accuracy: 0.8837 - loss: 0.3287 - val_accu
racy: 0.8775 - val_loss: 0.3119
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9189 - loss: 0.2351 - val_accu
racy: 0.8718 - val_loss: 0.3142
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9358 - loss: 0.1895 - val_accu
racy: 0.8876 - val_loss: 0.2766
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9498 - loss: 0.1511 - val_accu
racy: 0.8796 - val_loss: 0.3123
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9609 - loss: 0.1264 - val_accu
racy: 0.8830 - val_loss: 0.2959
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9669 - loss: 0.1087 - val_accu
racy: 0.8752 - val_loss: 0.3207
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9771 - loss: 0.0865 - val_accu
racy: 0.8516 - val_loss: 0.4625
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9773 - loss: 0.0767 - val_accu
racy: 0.8530 - val_loss: 0.4334
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9786 - loss: 0.0696 - val_accu
racy: 0.8791 - val_loss: 0.3675
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9866 - loss: 0.0504 - val_accu
racy: 0.8745 - val_loss: 0.4161
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9907 - loss: 0.0421 - val_accu
racy: 0.8485 - val_loss: 0.5069
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9921 - loss: 0.0390 - val_accu
racy: 0.8747 - val_loss: 0.4261
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9978 - loss: 0.0220 - val_accu
racy: 0.8738 - val_loss: 0.4487
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9985 - loss: 0.0170 - val_accu
racy: 0.8752 - val_loss: 0.4688
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9995 - loss: 0.0122 - val_accu
racy: 0.8743 - val_loss: 0.4945
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9962 - loss: 0.0204 - val_accu
racy: 0.8728 - val_loss: 0.5177
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9968 - loss: 0.0142 - val_accu
racy: 0.8726 - val_loss: 0.5342
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9999 - loss: 0.0058 - val_accu
```

```
racy: 0.8677 - val_loss: 0.5700
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9936 - loss: 0.0206 - val_accu
racy: 0.8719 - val_loss: 0.5780
```

In [39]:
```
history_dict32 = history32.history
history_dict32.keys()
```

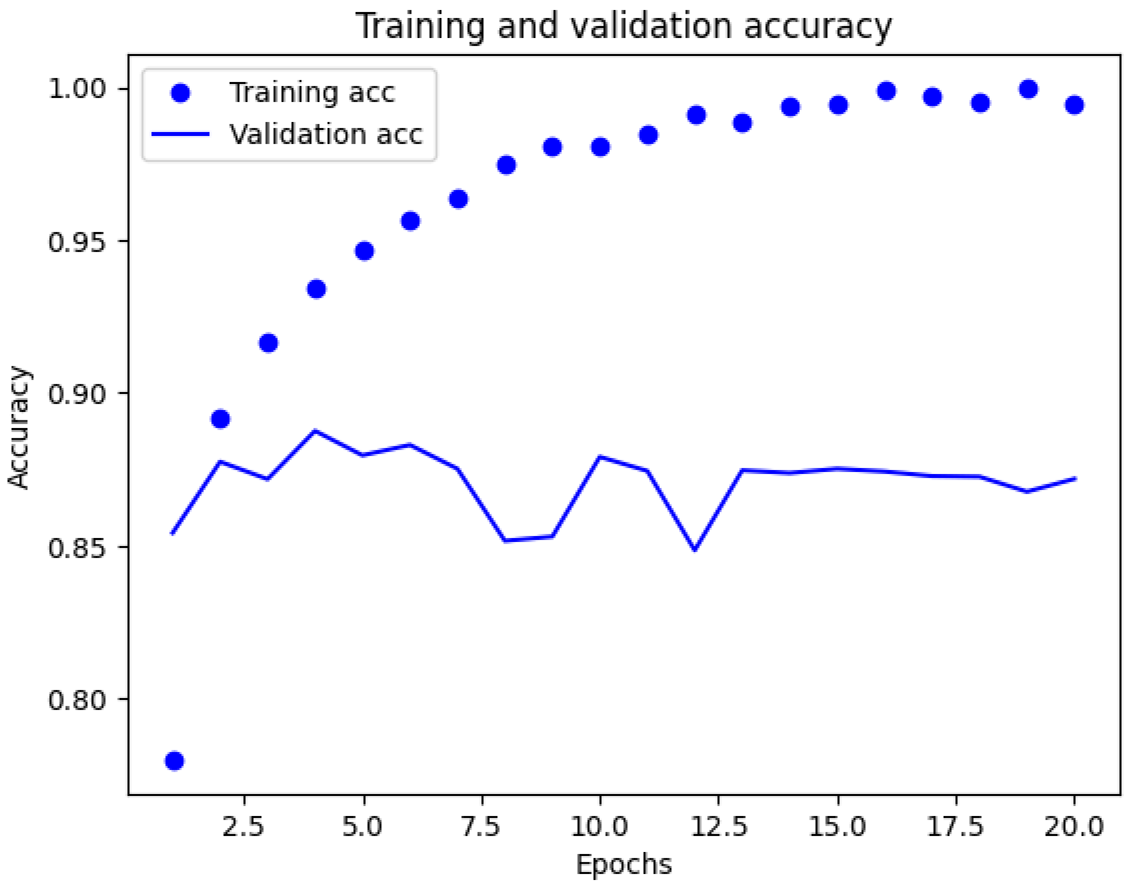
Out[39]:
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

In [40]:
```
loss_values = history_dict32["loss"]
val_loss_values = history_dict32["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()
```



In [41]:
```
plt.clf()
acc = history_dict32["accuracy"]
val_acc = history_dict32["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()
```



In [42]:
```
history_32 = model_32.fit(a_train, b_train, epochs=3, batch_size=512)
results_32 = model_32.evaluate(a_test, b_test)
results_32
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9470 - loss: 0.2091
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9662 - loss: 0.1107
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9771 - loss: 0.0786
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8658 - loss: 0.4170
```

Out[42]:
```
[0.41105881333351135, 0.8677999973297119]
```

In [43]:
```
model_32.predict(a_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
```

Out[43]:
```
array([[0.01664766],
       [0.99999994],
       [0.49116415],
       ...,
       [0.04295918],
       [0.03812066],
       [0.86407727]], dtype=float32)
```

the validation got an accuracy of 86.6%

training with 64 units

In [44]:
```
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
a_val = a_train[:10000]
partial_a_train = a_train[10000:]
b_val = b_train[:10000]
partial_b_train = b_train[10000:]
np.random.seed(123)
history64 = model_64.fit(partial_a_train,
partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
```

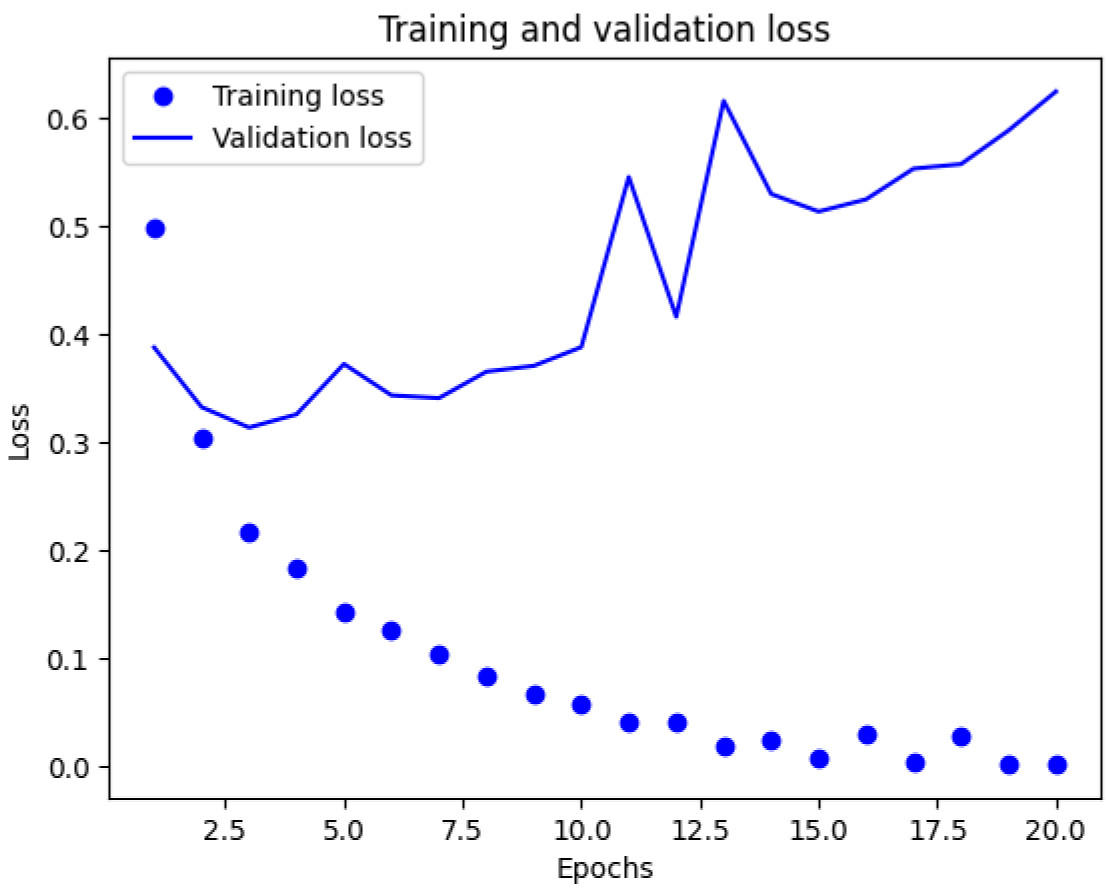
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 8s 217ms/step - accuracy: 0.7039 - loss: 0.5829 - val_acc
uracy: 0.8390 - val_loss: 0.3877
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.8756 - loss: 0.3222 - val_accu
racy: 0.8619 - val_loss: 0.3325
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9196 - loss: 0.2189 - val_accu
racy: 0.8731 - val_loss: 0.3135
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9254 - loss: 0.1939 - val_accu
racy: 0.8736 - val_loss: 0.3257
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9468 - loss: 0.1453 - val_accu
racy: 0.8595 - val_loss: 0.3724
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9518 - loss: 0.1329 - val_accu
racy: 0.8704 - val_loss: 0.3433
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9635 - loss: 0.1056 - val_accu
racy: 0.8740 - val_loss: 0.3408
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9779 - loss: 0.0740 - val_accu
racy: 0.8813 - val_loss: 0.3653
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9868 - loss: 0.0552 - val_accu
racy: 0.8778 - val_loss: 0.3706
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9896 - loss: 0.0447 - val_accu
racy: 0.8821 - val_loss: 0.3880
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9951 - loss: 0.0287 - val_accu
racy: 0.8578 - val_loss: 0.5454
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9915 - loss: 0.0315 - val_accu
racy: 0.8790 - val_loss: 0.4160
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9988 - loss: 0.0149 - val_accu
racy: 0.8368 - val_loss: 0.6158
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9950 - loss: 0.0220 - val_accu
racy: 0.8704 - val_loss: 0.5297
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9995 - loss: 0.0103 - val_accu
racy: 0.8771 - val_loss: 0.5134
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9955 - loss: 0.0177 - val_accu
racy: 0.8785 - val_loss: 0.5247
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9999 - loss: 0.0042 - val_accu
racy: 0.8762 - val_loss: 0.5531
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9969 - loss: 0.0120 - val_accu
racy: 0.8778 - val_loss: 0.5573
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9999 - loss: 0.0027 - val_accu
```

```
racy: 0.8772 - val_loss: 0.5884
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 1.0000 - loss: 0.0019 - val_accu
racy: 0.8749 - val_loss: 0.6245
```

In [45]:
```
history_dict64 = history64.history
history_dict64.keys()
```

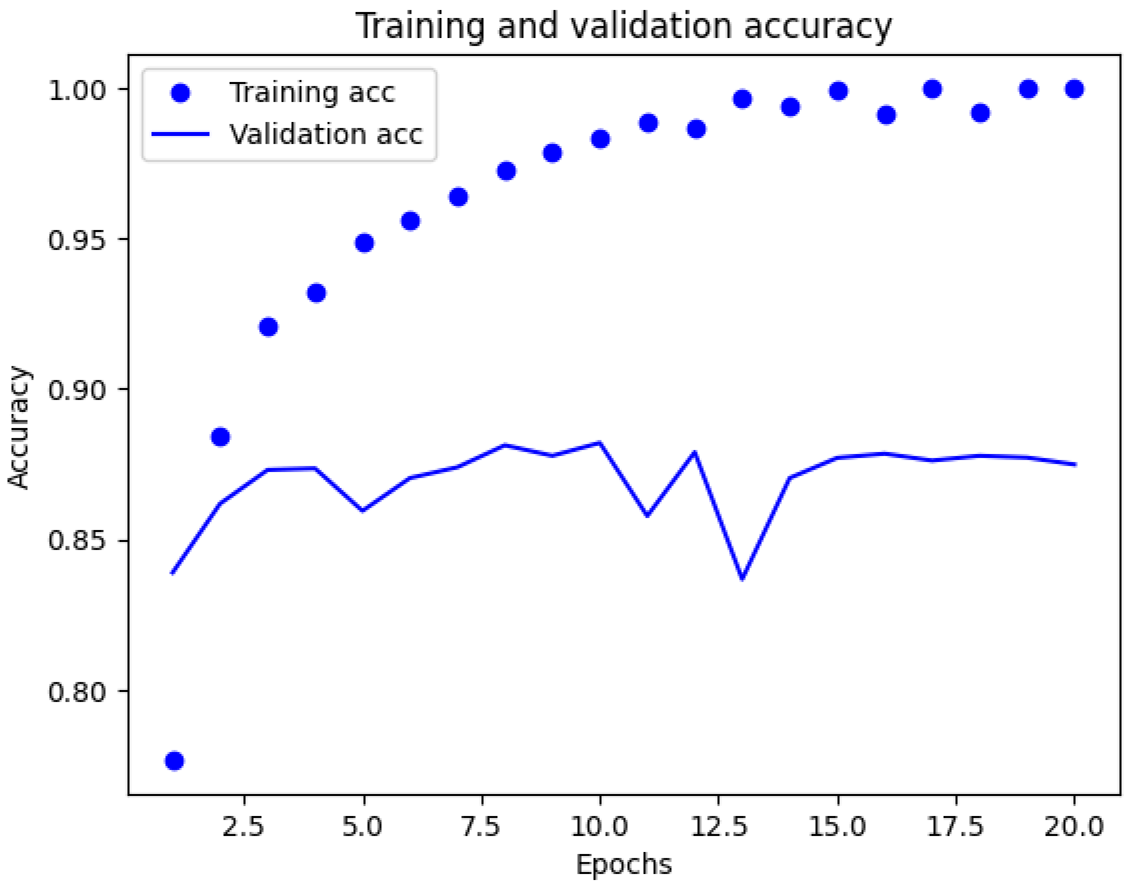
Out[45]:
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

In [46]:
```
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()
```



In [47]:
```
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()
```



In [48]:
```
history_64 = model_64.fit(a_train, b_train, epochs=3, batch_size=512)
results_64 = model_64.evaluate(a_test, b_test)
results_64
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9389 - loss: 0.2373
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9693 - loss: 0.0985
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9835 - loss: 0.0587
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8686 - loss: 0.4194
```

Out[48]:
```
[0.41748759150505066, 0.8703600168228149]
```

In [49]:
```
model_64.predict(a_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```

Out[49]:
```
array([[0.02021047],
       [0.9999997 ],
       [0.4983846 ],
       ...,
       [0.01497949],
       [0.00630524],
       [0.6154616 ]], dtype=float32)
```

validation has accuracy 85.18%

Training the model with 128 units

In [50]:
```
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
a_val = a_train[:10000]
partial_a_train = a_train[10000:]
b_val = b_train[:10000]
partial_b_train = b_train[10000:]
np.random.seed(123)
history128 = model_128.fit(partial_a_train,
partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 8s 212ms/step - accuracy: 0.6503 - loss: 0.6060 - val_acc
uracy: 0.8499 - val_loss: 0.3661
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.8747 - loss: 0.3147 - val_accu
racy: 0.8779 - val_loss: 0.2985
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9095 - loss: 0.2319 - val_accu
racy: 0.8637 - val_loss: 0.3258
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9377 - loss: 0.1698 - val_accu
racy: 0.8702 - val_loss: 0.3281
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9492 - loss: 0.1406 - val_accu
racy: 0.8859 - val_loss: 0.2900
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9611 - loss: 0.1152 - val_accu
racy: 0.8762 - val_loss: 0.3169
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.9717 - loss: 0.0867 - val_accu
racy: 0.8509 - val_loss: 0.4451
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9652 - loss: 0.0913 - val_accu
racy: 0.8654 - val_loss: 0.4297
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9820 - loss: 0.0528 - val_accu
racy: 0.8784 - val_loss: 0.4033
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9911 - loss: 0.0337 - val_accu
racy: 0.8800 - val_loss: 0.3916
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9987 - loss: 0.0131 - val_accu
racy: 0.8698 - val_loss: 0.5185
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9649 - loss: 0.0996 - val_accu
racy: 0.8790 - val_loss: 0.4591
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9995 - loss: 0.0060 - val_accu
racy: 0.7880 - val_loss: 1.1213
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9647 - loss: 0.1170 - val_accu
racy: 0.8779 - val_loss: 0.5031
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 1.0000 - loss: 0.0030 - val_accu
racy: 0.8694 - val_loss: 0.5854
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9916 - loss: 0.0366 - val_accu
racy: 0.8773 - val_loss: 0.5102
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 0.0025 - val_accu
racy: 0.8799 - val_loss: 0.5595
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 0.0015 - val_accu
racy: 0.8743 - val_loss: 0.6206
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9970 - loss: 0.0134 - val_accu
```

```
racy: 0.8757 - val_loss: 0.5595
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 0.0017 - val_accu
racy: 0.8781 - val_loss: 0.5754
```

In [51]:

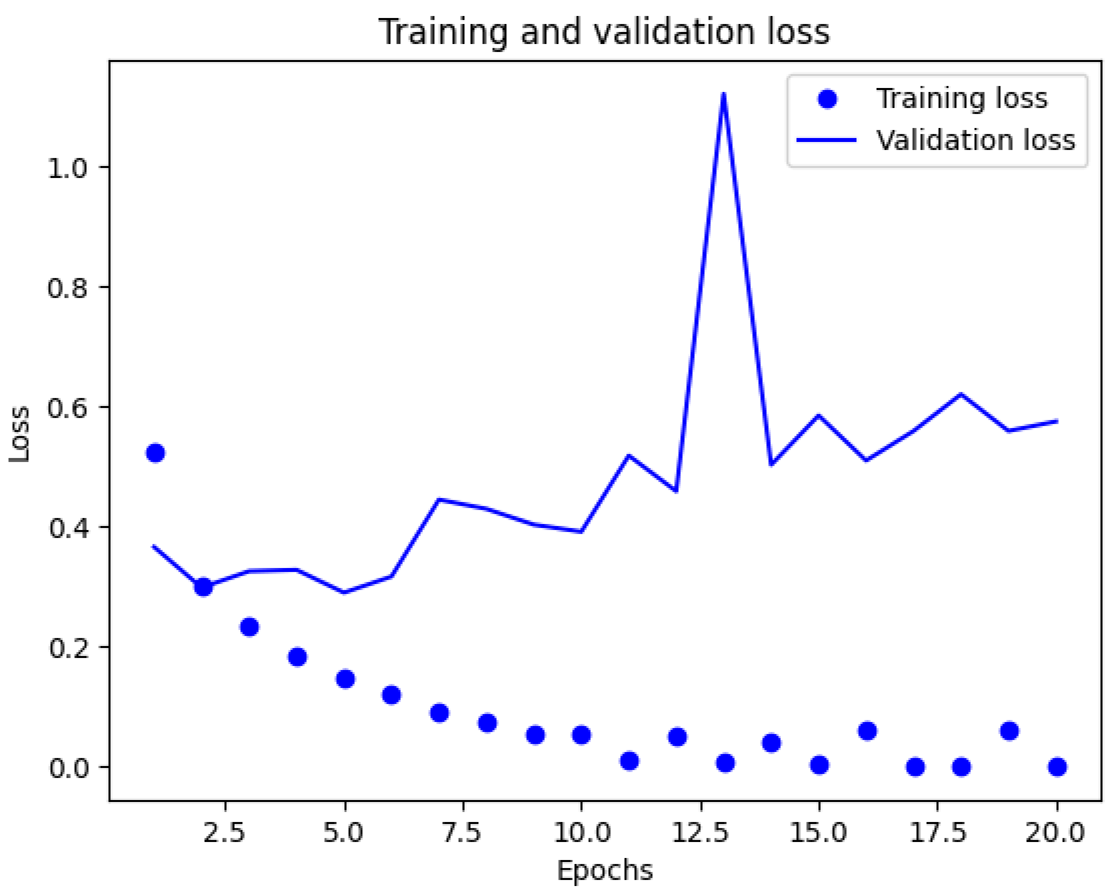
```
history_dict128 = history128.history
history_dict128.keys()
```

Out[51]:

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

In [52]:

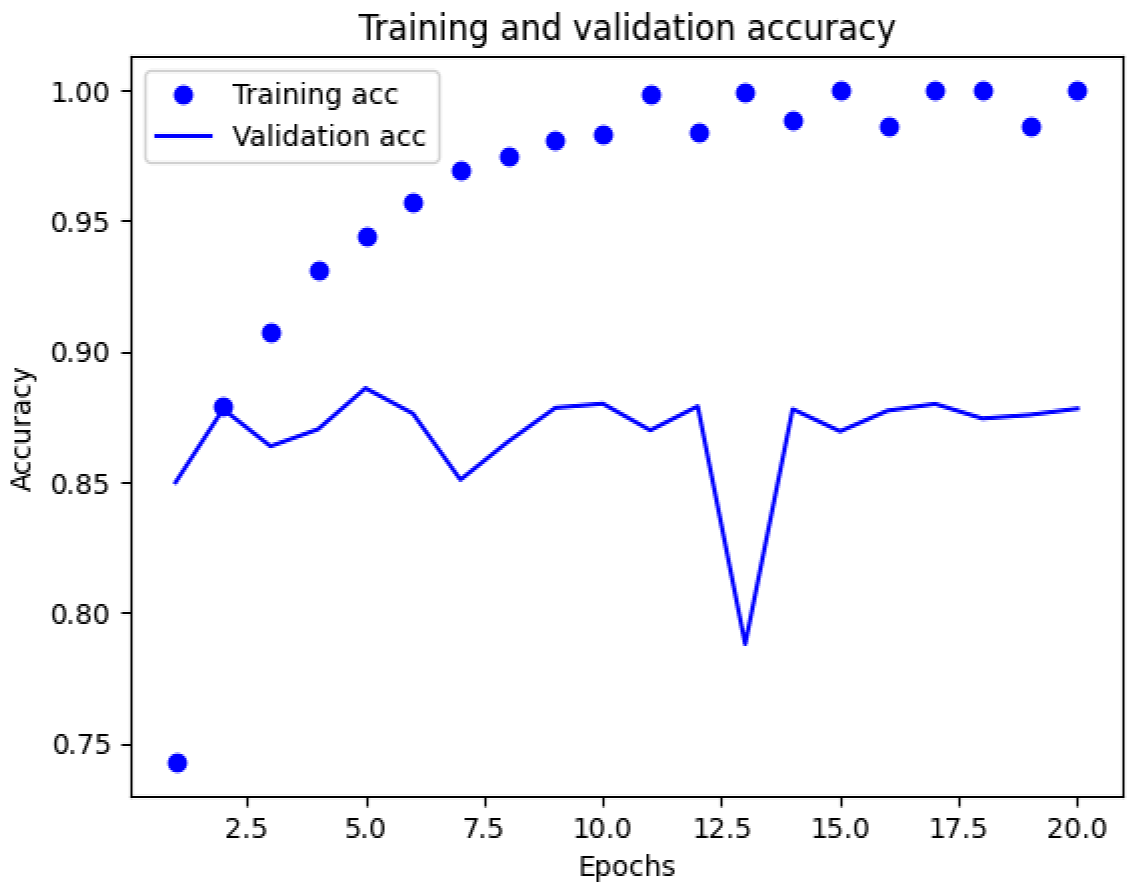
```
loss_values = history_dict128["loss"]
val_loss_values = history_dict128["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")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



In [53]:

```
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()
```



In [54]:
```
history_128 = model_128.fit(a_train, b_train, epochs=2, batch_size=512)
results_128 = model_128.evaluate(a_test, b_test)
results_128
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9502 - loss: 0.1732
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9751 - loss: 0.0816
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8672 - loss: 0.3730
```

Out[54]:
```
[0.3680635094642639, 0.8704400062561035]
```

In [55]:
```
model_128.predict(a_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```

Out[55]:
```
array([[0.01518019],
       [0.99999994],
       [0.82240325],
       ...,
       [0.04891635],
       [0.00815896],
       [0.6977612 ]], dtype=float32)
```

accuracy of 96.7% and loss of 0.37

MSE LOSS_FUNCTION

In [56]:
```
np.random.seed(123)
model_MSE = keras.Sequential([
layers.Dense(16, activation="relu"),
layers.Dense(16, activation="relu"),
layers.Dense(1, activation="sigmoid")
])

model_MSE.compile(optimizer="rmsprop",
loss="mse",
metrics=["accuracy"])

a_val = a_train[:10000]
partial_a_train = a_train[10000:]
b_val = b_train[:10000]
partial_b_train = b_train[10000:]

np.random.seed(123)
history_model_MSE = model_MSE.fit(partial_a_train,
partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
```

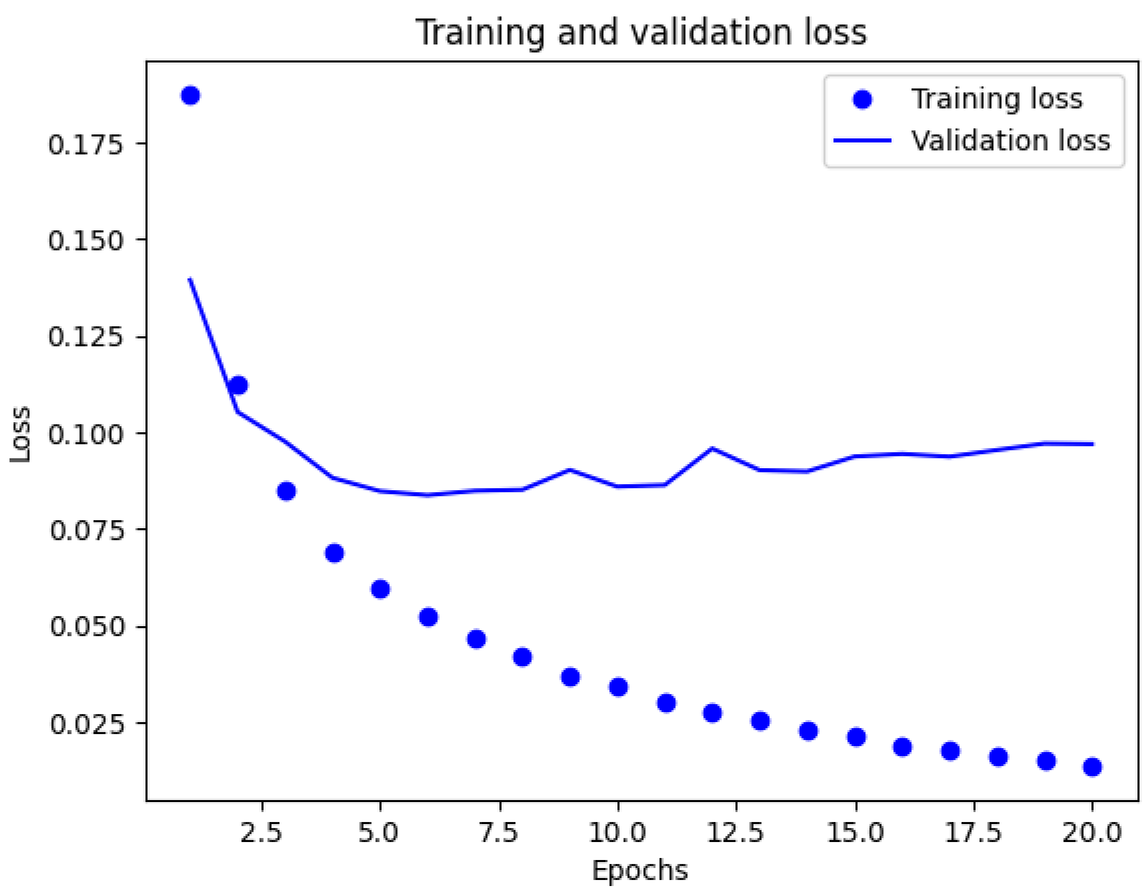
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 150ms/step - accuracy: 0.6829 - loss: 0.2158 - val_acc
uracy: 0.8509 - val_loss: 0.1394
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.8830 - loss: 0.1183 - val_accu
racy: 0.8786 - val_loss: 0.1052
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9068 - loss: 0.0874 - val_accu
racy: 0.8753 - val_loss: 0.0976
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9246 - loss: 0.0702 - val_accu
racy: 0.8847 - val_loss: 0.0882
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9348 - loss: 0.0596 - val_accu
racy: 0.8868 - val_loss: 0.0847
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9406 - loss: 0.0533 - val_accu
racy: 0.8864 - val_loss: 0.0837
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9531 - loss: 0.0452 - val_accu
racy: 0.8860 - val_loss: 0.0848
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9571 - loss: 0.0414 - val_accu
racy: 0.8834 - val_loss: 0.0851
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9643 - loss: 0.0359 - val_accu
racy: 0.8807 - val_loss: 0.0902
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9705 - loss: 0.0330 - val_accu
racy: 0.8800 - val_loss: 0.0859
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9718 - loss: 0.0301 - val_accu
racy: 0.8824 - val_loss: 0.0863
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9768 - loss: 0.0261 - val_accu
racy: 0.8696 - val_loss: 0.0958
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9754 - loss: 0.0260 - val_accu
racy: 0.8789 - val_loss: 0.0902
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9824 - loss: 0.0227 - val_accu
racy: 0.8786 - val_loss: 0.0898
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9835 - loss: 0.0205 - val_accu
racy: 0.8776 - val_loss: 0.0937
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9852 - loss: 0.0184 - val_accu
racy: 0.8728 - val_loss: 0.0944
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9861 - loss: 0.0179 - val_accu
racy: 0.8770 - val_loss: 0.0937
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9891 - loss: 0.0151 - val_accu
racy: 0.8757 - val_loss: 0.0954
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9900 - loss: 0.0137 - val_accu
```

```
racy: 0.8725 - val_loss: 0.0971
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9923 - loss: 0.0108 - val_accu
racy: 0.8734 - val_loss: 0.0969
```

In [57]:
```
history_dict_MSE = history_model_MSE.history
history_dict_MSE.keys()
```

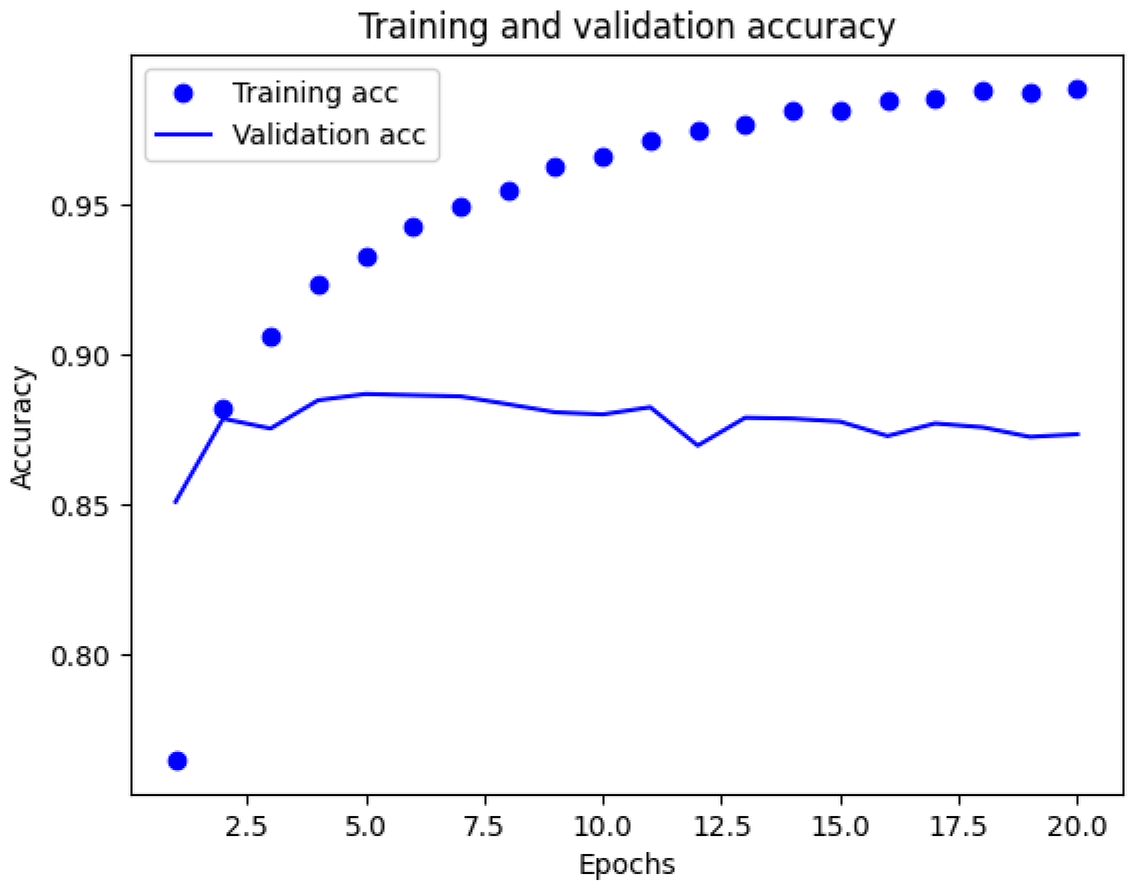
Out[57]:
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

In [58]:
```
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()
```



In [59]:
```
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()
```



In [60]:
```
model_MSE.fit(a_train, b_train, epochs=8, batch_size=512)
results_MSE = model_MSE.evaluate(a_test, b_test)
results_MSE
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9431 - loss: 0.0465
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9553 - loss: 0.0389
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9640 - loss: 0.0333
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9687 - loss: 0.0303
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9721 - loss: 0.0270
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9750 - loss: 0.0254
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9777 - loss: 0.0232
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9805 - loss: 0.0205
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8603 - loss: 0.1121
```

Out[60]:
```
[0.10940994322299957, 0.8639600276947021]
```

In [61]:
```
model_MSE.predict(a_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```

Out[61]:
```
array([[0.04115713],
       [1.        ],
       [0.99288017],
       ...,
       [0.13062851],
       [0.02381671],
       [0.9707612 ]], dtype=float32)
```

Tanh ACtivation function

In [62]:
```
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'])
a_val = a_train[:10000]
partial_a_train = a_train[10000:]
b_val = b_train[:10000]
partial_b_train = b_train[10000:]
np.random.seed(123)
history_tanh = model_tanh.fit(partial_a_train,
partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
```

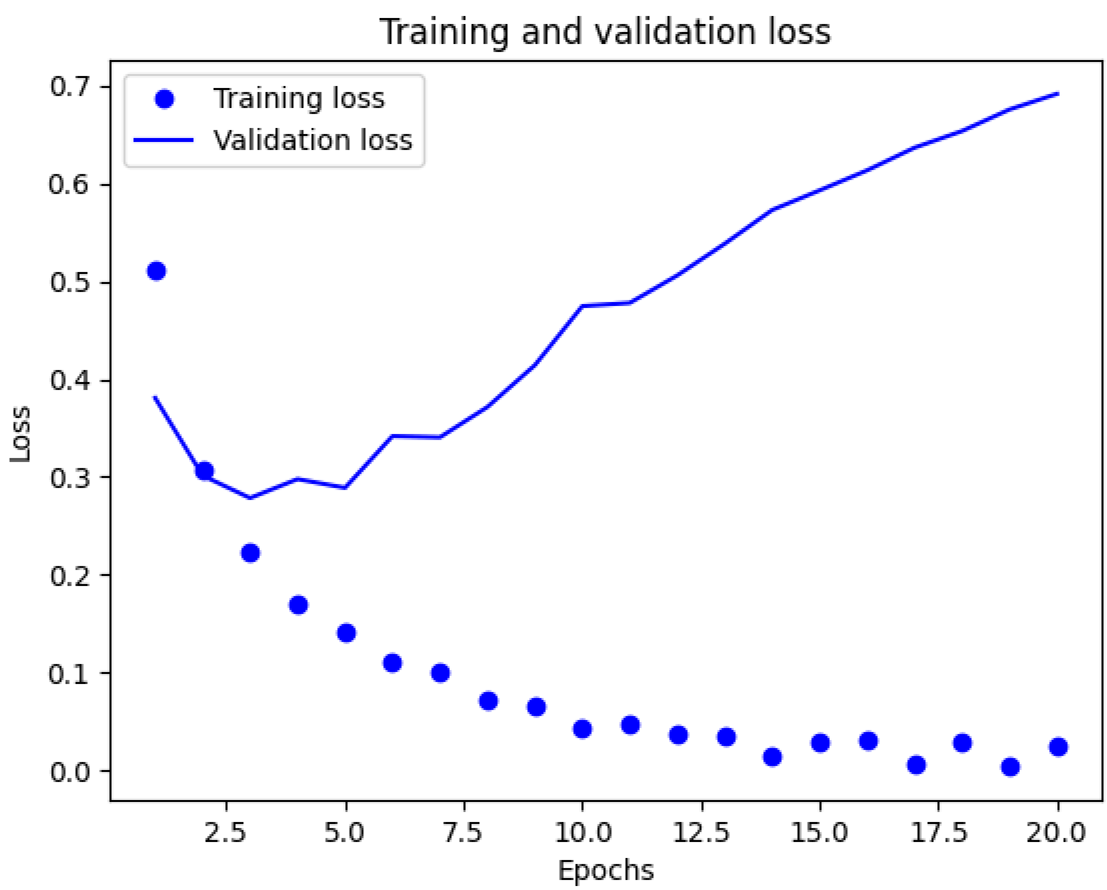
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 10s 261ms/step - accuracy: 0.6930 - loss: 0.5898 - val_ac
curacy: 0.8622 - val_loss: 0.3807
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.8925 - loss: 0.3252 - val_accu
racy: 0.8816 - val_loss: 0.3015
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9206 - loss: 0.2284 - val_accu
racy: 0.8869 - val_loss: 0.2785
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9421 - loss: 0.1748 - val_accu
racy: 0.8792 - val_loss: 0.2978
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9498 - loss: 0.1450 - val_accu
racy: 0.8850 - val_loss: 0.2889
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9653 - loss: 0.1078 - val_accu
racy: 0.8736 - val_loss: 0.3417
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9725 - loss: 0.0889 - val_accu
racy: 0.8782 - val_loss: 0.3405
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9821 - loss: 0.0671 - val_accu
racy: 0.8788 - val_loss: 0.3714
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9829 - loss: 0.0591 - val_accu
racy: 0.8714 - val_loss: 0.4141
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9905 - loss: 0.0428 - val_accu
racy: 0.8642 - val_loss: 0.4744
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9913 - loss: 0.0351 - val_accu
racy: 0.8714 - val_loss: 0.4777
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9938 - loss: 0.0297 - val_accu
racy: 0.8699 - val_loss: 0.5058
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9961 - loss: 0.0211 - val_accu
racy: 0.8697 - val_loss: 0.5381
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9988 - loss: 0.0135 - val_accu
racy: 0.8666 - val_loss: 0.5726
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9880 - loss: 0.0395 - val_accu
racy: 0.8665 - val_loss: 0.5926
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9963 - loss: 0.0163 - val_accu
racy: 0.8670 - val_loss: 0.6130
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9995 - loss: 0.0066 - val_accu
racy: 0.8665 - val_loss: 0.6362
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9946 - loss: 0.0208 - val_accu
racy: 0.8658 - val_loss: 0.6532
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9996 - loss: 0.0048 - val_accu
```

```
racy: 0.8665 - val_loss: 0.6752
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9967 - loss: 0.0126 - val_accu
racy: 0.8651 - val_loss: 0.6912
```

In [63]:
```python
history_dict_tanh = history_tanh.history
history_dict_tanh.keys()
```

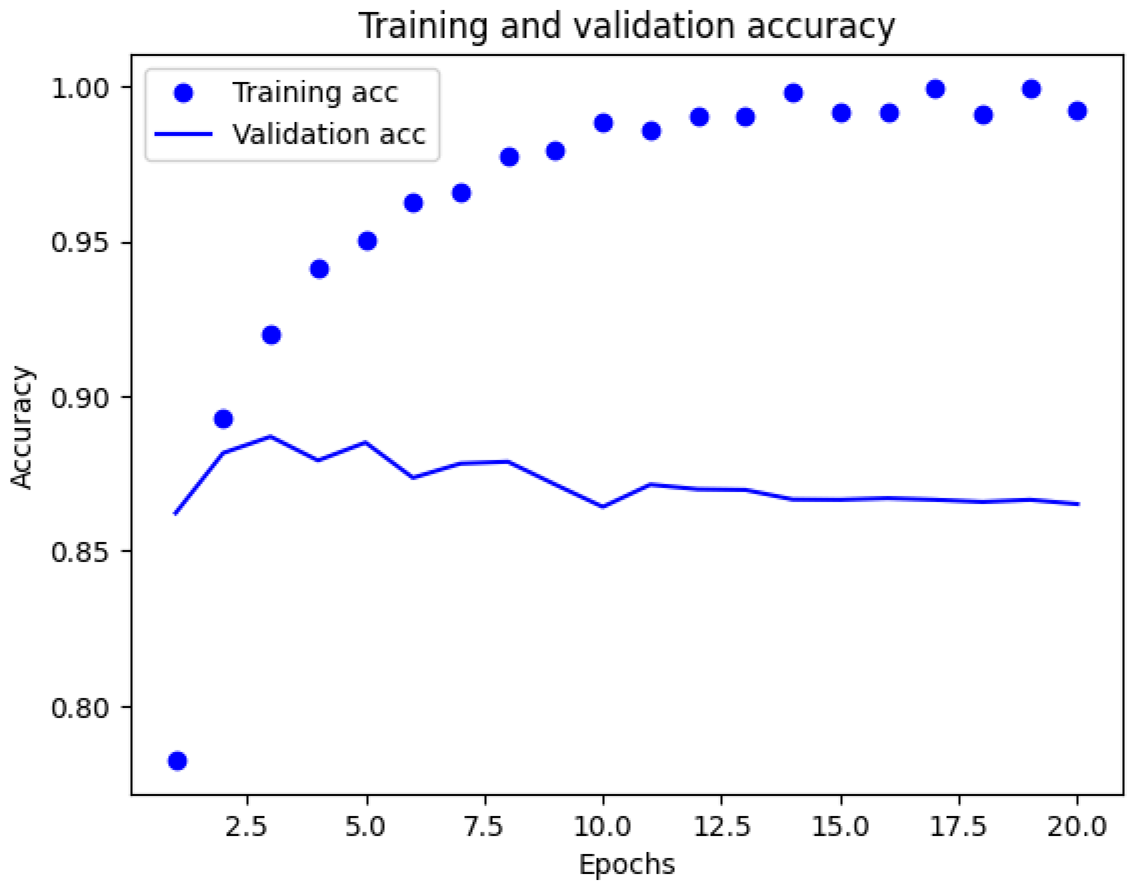
Out[63]:
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

In [64]:
```python
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()
```



In [65]:
```python
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()
```



In [66]:
```
model_tanh.fit(a_train, b_train, epochs=8, batch_size=512)
results_tanh = model_tanh.evaluate(a_test, b_test)
results_tanh
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.9443 - loss: 0.2708
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9615 - loss: 0.1374
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9690 - loss: 0.1054
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9710 - loss: 0.0924
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9797 - loss: 0.0740
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9823 - loss: 0.0635
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9820 - loss: 0.0599
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9846 - loss: 0.0558
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8518 - loss: 0.6175
```

Out[66]: [0.6067667007446289, 0.8536800146102905]

ADAM OPTIMIZER FUNCTION

In [67]:
```
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'])
a_val = a_train[:10000]
partial_a_train = a_train[10000:]
b_val = b_train[:10000]
partial_b_train = b_train[10000:]
np.random.seed(123)
history_adam = model_adam.fit(partial_a_train,
partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
```

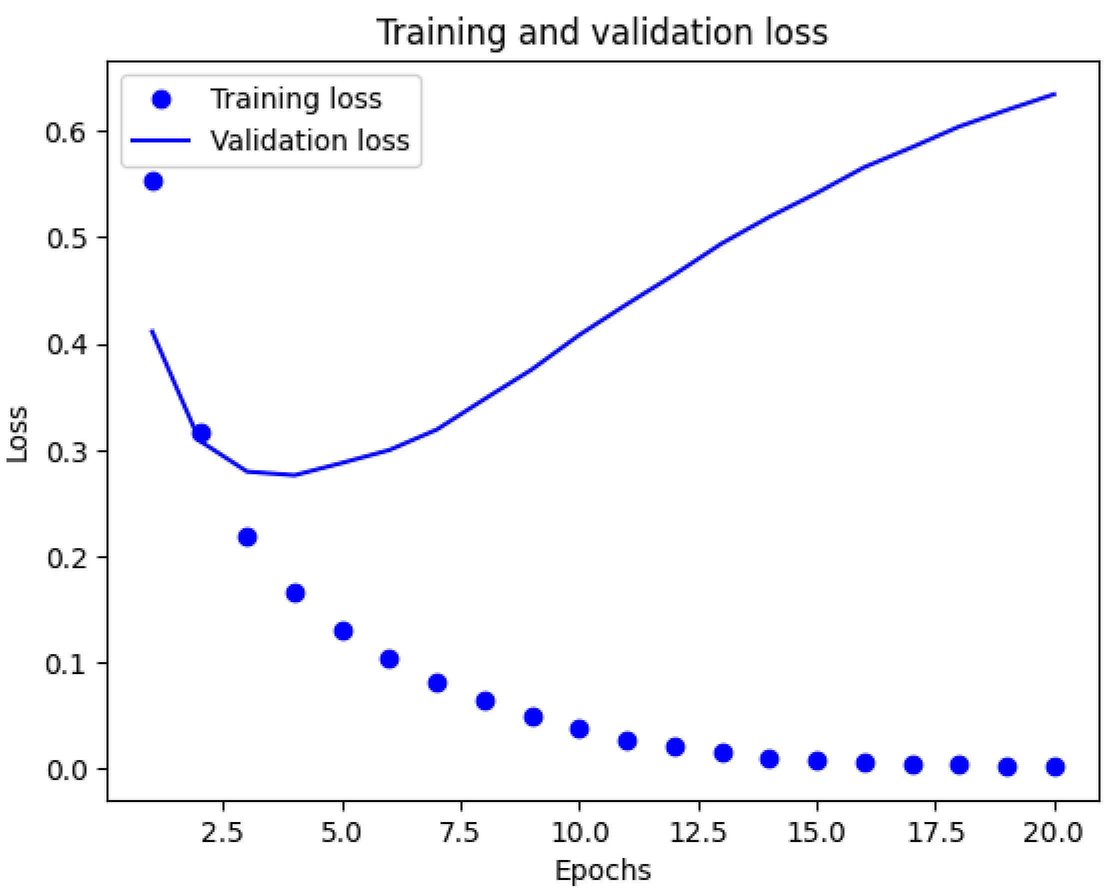
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 8s 164ms/step - accuracy: 0.6869 - loss: 0.6245 - val_acc
uracy: 0.8551 - val_loss: 0.4113
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.8924 - loss: 0.3399 - val_accu
racy: 0.8829 - val_loss: 0.3086
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9300 - loss: 0.2272 - val_accu
racy: 0.8893 - val_loss: 0.2795
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9503 - loss: 0.1689 - val_accu
racy: 0.8880 - val_loss: 0.2761
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9635 - loss: 0.1320 - val_accu
racy: 0.8853 - val_loss: 0.2877
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9747 - loss: 0.1024 - val_accu
racy: 0.8844 - val_loss: 0.2998
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9814 - loss: 0.0839 - val_accu
racy: 0.8799 - val_loss: 0.3191
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9869 - loss: 0.0655 - val_accu
racy: 0.8802 - val_loss: 0.3477
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9928 - loss: 0.0489 - val_accu
racy: 0.8773 - val_loss: 0.3756
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9950 - loss: 0.0364 - val_accu
racy: 0.8765 - val_loss: 0.4080
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9965 - loss: 0.0274 - val_accu
racy: 0.8737 - val_loss: 0.4369
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9985 - loss: 0.0215 - val_accu
racy: 0.8740 - val_loss: 0.4647
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9995 - loss: 0.0149 - val_accu
racy: 0.8719 - val_loss: 0.4941
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9998 - loss: 0.0112 - val_accu
racy: 0.8716 - val_loss: 0.5190
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9999 - loss: 0.0081 - val_accu
racy: 0.8708 - val_loss: 0.5415
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 1.0000 - loss: 0.0065 - val_accu
racy: 0.8697 - val_loss: 0.5657
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 1.0000 - loss: 0.0048 - val_accu
racy: 0.8691 - val_loss: 0.5844
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 1.0000 - loss: 0.0040 - val_accu
racy: 0.8688 - val_loss: 0.6040
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 1.0000 - loss: 0.0035 - val_accu
```

```
racy: 0.8695 - val_loss: 0.6195
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accu
racy: 0.8698 - val_loss: 0.6344
```

In [68]:
```
history_dict_adam = history_adam.history
history_dict_adam.keys()
```

Out[68]: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])

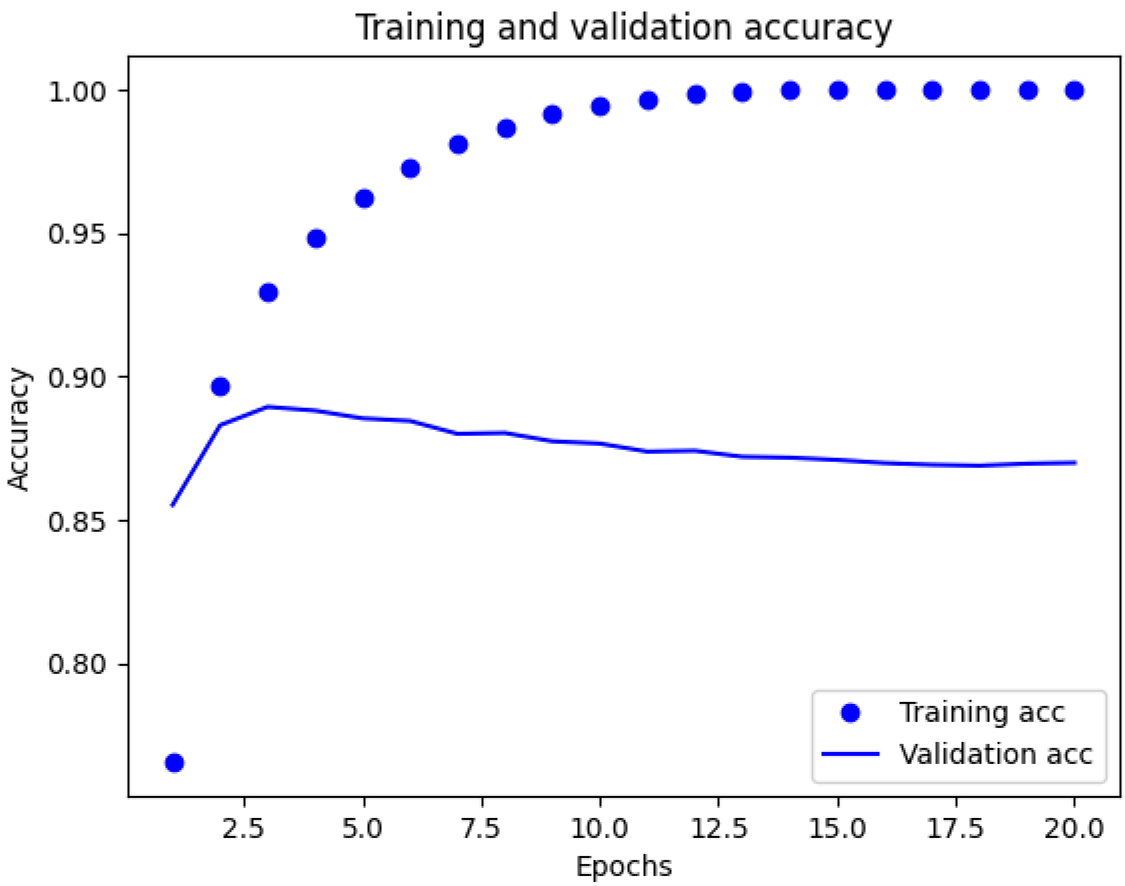
In [69]:
```
loss_values = history_dict_adam["loss"]
val_loss_values = history_dict_adam["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "bo", label="Training loss")
plt.plot(epochs, val_loss_values, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
44
```



Out[69]: 44

In [70]:
```
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()
```



In [71]:
```
model_adam.fit(a_train, b_train, epochs=4, batch_size=512)
results_adam = model_adam.evaluate(a_test, b_test)
results_adam
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9471 - loss: 0.2269
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9722 - loss: 0.0944
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9860 - loss: 0.0549
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9938 - loss: 0.0361
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8576 - loss: 0.5169
```

Out[71]: [0.5144559144973755, 0.8578400015830994]

REGULARIZATION

In [72]:
```
from tensorflow.keras import regularizers
np.random.seed(123)
model_regularization = keras.Sequential([
layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.001)),
```

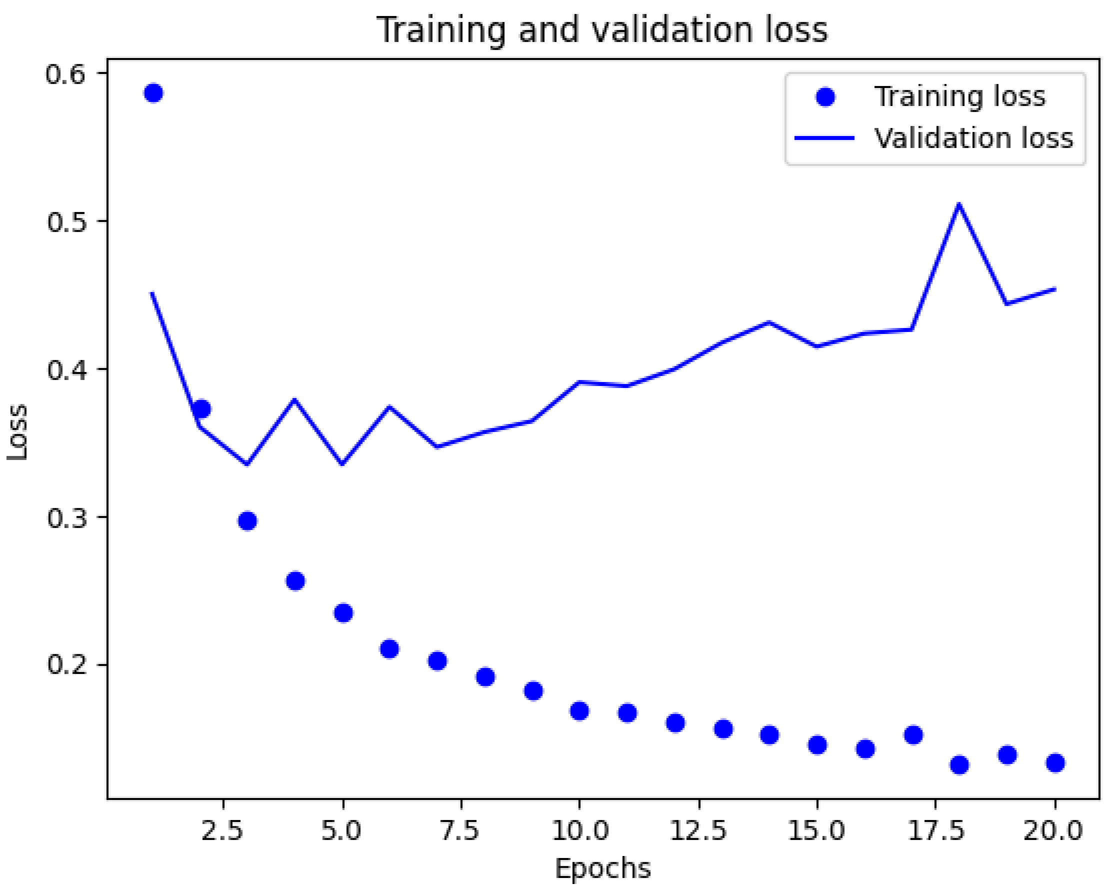
```
layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.001)),
layers.Dense(1, activation="sigmoid")
])
model_regularization.compile(optimizer="rmsprop",
loss="binary_crossentropy",
metrics=["accuracy"])
np.random.seed(123)
history_model_regularization = model_regularization.fit(partial_a_train,
partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
history_dict_regularization = history_model_regularization.history
history_dict_regularization.keys()
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 10s 180ms/step - accuracy: 0.6608 - loss: 0.6643 - val_ac
curacy: 0.8577 - val_loss: 0.4502
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.8873 - loss: 0.3959 - val_accu
racy: 0.8819 - val_loss: 0.3602
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9184 - loss: 0.2995 - val_accu
racy: 0.8881 - val_loss: 0.3346
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9341 - loss: 0.2541 - val_accu
racy: 0.8643 - val_loss: 0.3787
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9415 - loss: 0.2299 - val_accu
racy: 0.8851 - val_loss: 0.3347
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9521 - loss: 0.2054 - val_accu
racy: 0.8758 - val_loss: 0.3737
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9557 - loss: 0.1937 - val_accu
racy: 0.8838 - val_loss: 0.3465
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9653 - loss: 0.1804 - val_accu
racy: 0.8816 - val_loss: 0.3567
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9696 - loss: 0.1678 - val_accu
racy: 0.8815 - val_loss: 0.3641
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9719 - loss: 0.1603 - val_accu
racy: 0.8779 - val_loss: 0.3905
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9717 - loss: 0.1623 - val_accu
racy: 0.8793 - val_loss: 0.3878
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9760 - loss: 0.1527 - val_accu
racy: 0.8785 - val_loss: 0.3993
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9776 - loss: 0.1460 - val_accu
racy: 0.8752 - val_loss: 0.4172
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9792 - loss: 0.1423 - val_accu
racy: 0.8678 - val_loss: 0.4309
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9814 - loss: 0.1371 - val_accu
racy: 0.8760 - val_loss: 0.4145
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9804 - loss: 0.1377 - val_accu
racy: 0.8742 - val_loss: 0.4234
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9788 - loss: 0.1401 - val_accu
racy: 0.8744 - val_loss: 0.4261
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9883 - loss: 0.1215 - val_accu
racy: 0.8526 - val_loss: 0.5111
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9794 - loss: 0.1367 - val_accu
```

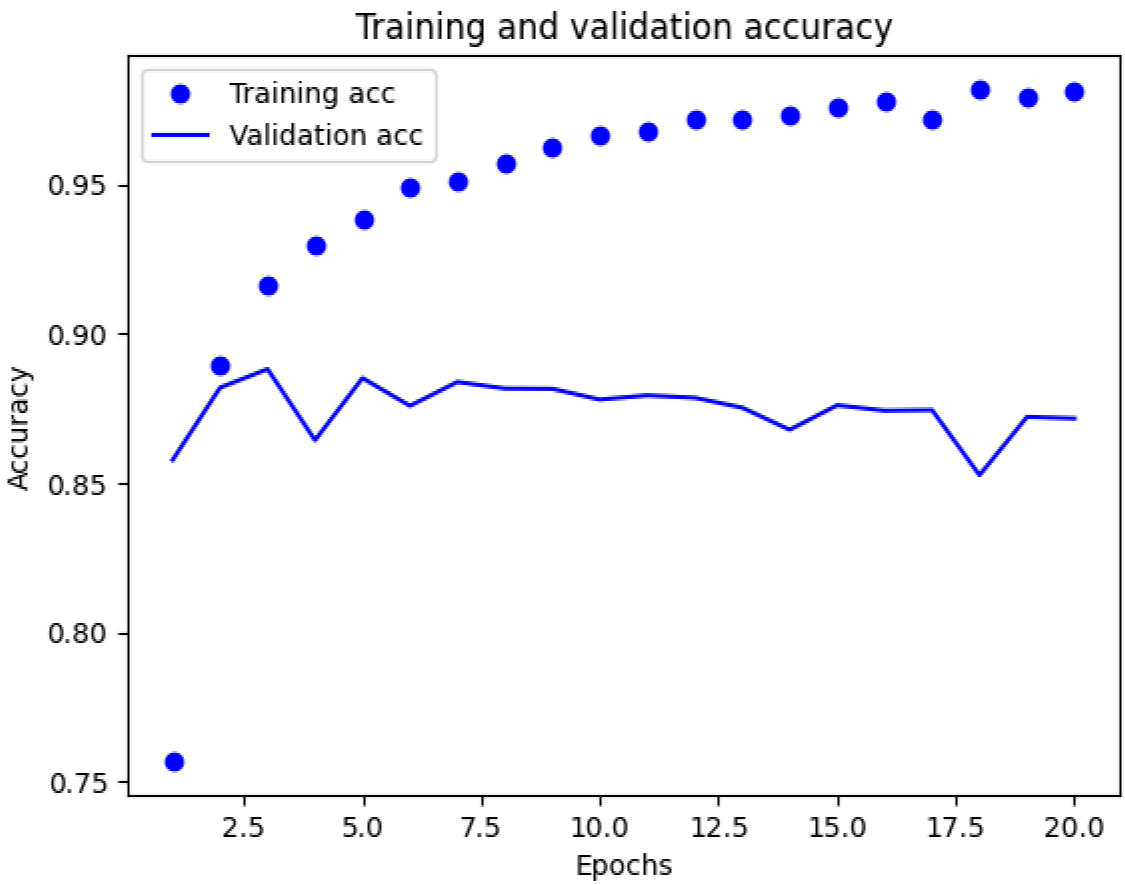
```
racy: 0.8721 - val_loss: 0.4434
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9863 - loss: 0.1236 - val_accu
racy: 0.8716 - val_loss: 0.4533
```

Out[72]: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])

In [73]:
```
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()
```



In [74]:
```
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()
```



In [75]:
```
model_regularization.fit(a_train, b_train, epochs=8, batch_size=512)
results_regularization = model_regularization.evaluate(a_test, b_test)
results_regularization
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9385 - loss: 0.2580
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9543 - loss: 0.2020
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9596 - loss: 0.1799
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9625 - loss: 0.1731
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9591 - loss: 0.1725
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9665 - loss: 0.1639
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9721 - loss: 0.1523
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9770 - loss: 0.1429
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8620 - loss: 0.4561
```

Out[75]:
```
[0.4544123709201813, 0.8629999756813049]
```

In [77]:
```
#### Loss 0.43 is and accuracy is 87.03%
```

DROPOUT

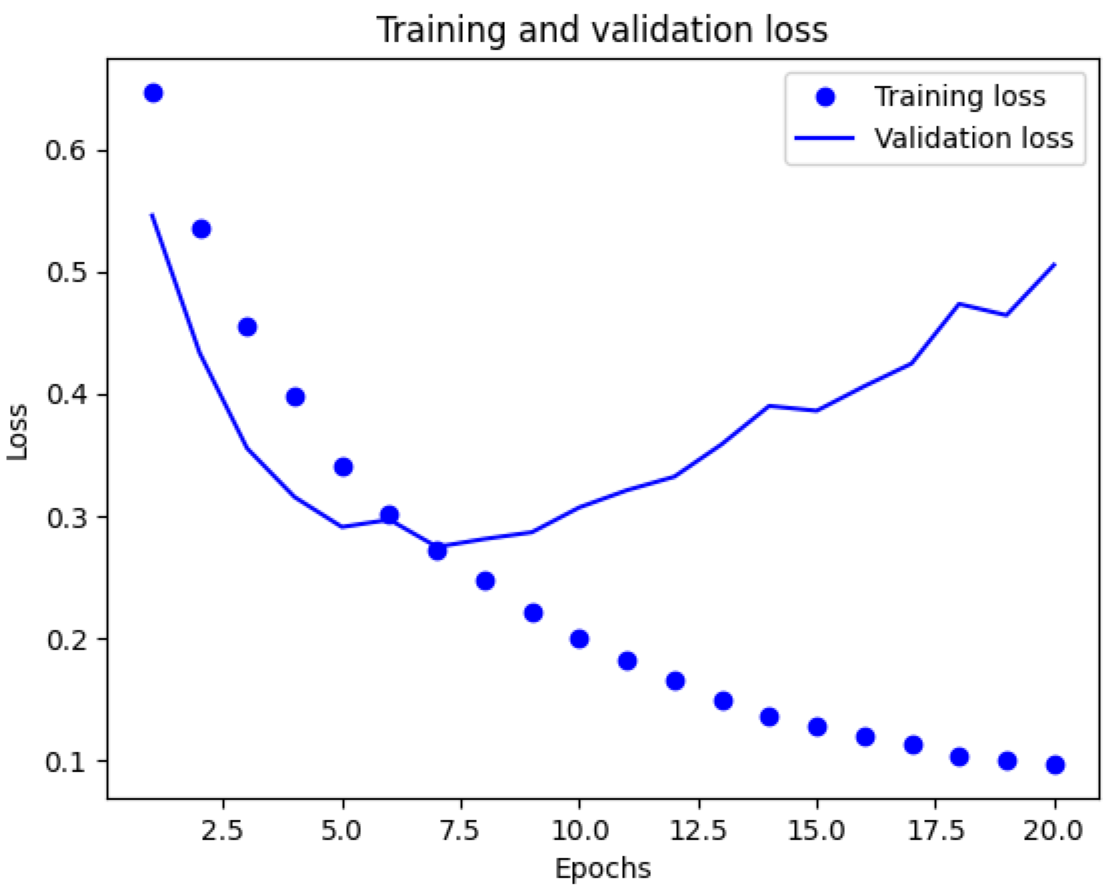
In [78]:
```
from tensorflow.keras import regularizers
np.random.seed(123)
model_Dropout = keras.Sequential([
layers.Dense(16, activation="relu"),
layers.Dropout(0.5),
layers.Dense(16, activation="relu"),
layers.Dropout(0.5),
layers.Dense(1, activation="sigmoid")
])
model_Dropout.compile(optimizer="rmsprop",
loss="binary_crossentropy",
metrics=["accuracy"])
np.random.seed(123)
history_model_Dropout = model_Dropout.fit(partial_a_train,
partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
history_dict_Dropout = history_model_Dropout.history
history_dict_Dropout.keys()
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 125ms/step - accuracy: 0.5654 - loss: 0.6721 - val_acc
uracy: 0.8016 - val_loss: 0.5457
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.7296 - loss: 0.5555 - val_accu
racy: 0.8675 - val_loss: 0.4333
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.7964 - loss: 0.4683 - val_accu
racy: 0.8801 - val_loss: 0.3552
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8341 - loss: 0.4108 - val_accu
racy: 0.8847 - val_loss: 0.3153
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.8638 - loss: 0.3491 - val_accu
racy: 0.8872 - val_loss: 0.2908
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.8821 - loss: 0.3047 - val_accu
racy: 0.8843 - val_loss: 0.2968
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.8989 - loss: 0.2768 - val_accu
racy: 0.8890 - val_loss: 0.2745
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9130 - loss: 0.2470 - val_accu
racy: 0.8905 - val_loss: 0.2810
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9209 - loss: 0.2161 - val_accu
racy: 0.8898 - val_loss: 0.2865
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9289 - loss: 0.1998 - val_accu
racy: 0.8883 - val_loss: 0.3068
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9331 - loss: 0.1814 - val_accu
racy: 0.8846 - val_loss: 0.3208
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9423 - loss: 0.1659 - val_accu
racy: 0.8889 - val_loss: 0.3319
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9491 - loss: 0.1450 - val_accu
racy: 0.8829 - val_loss: 0.3585
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9500 - loss: 0.1344 - val_accu
racy: 0.8872 - val_loss: 0.3898
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9543 - loss: 0.1245 - val_accu
racy: 0.8872 - val_loss: 0.3859
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9535 - loss: 0.1221 - val_accu
racy: 0.8872 - val_loss: 0.4059
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9596 - loss: 0.1116 - val_accu
racy: 0.8878 - val_loss: 0.4245
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9598 - loss: 0.1051 - val_accu
racy: 0.8863 - val_loss: 0.4733
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9618 - loss: 0.1019 - val_accu
```

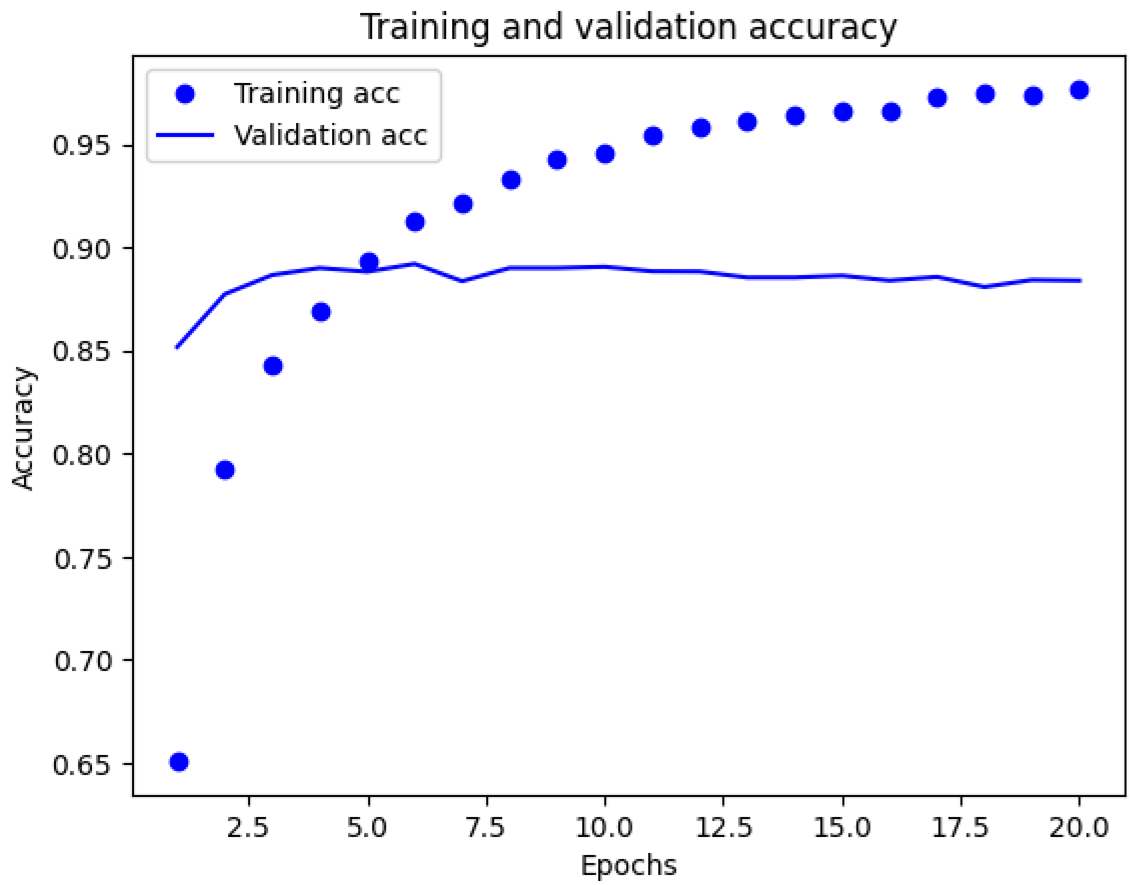
```
racy: 0.8855 - val_loss: 0.4641
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9608 - loss: 0.0982 - val_accu
racy: 0.8859 - val_loss: 0.5052
```

Out[78]: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])

In [79]:
```
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()
```



In [149…
```
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()
```



In [150…

```
model_Dropout.fit(a_train, b_train, epochs=8, batch_size=512)
results_Dropout = model_Dropout.evaluate(a_test, b_test)
results_Dropout
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9268 - loss: 0.2548
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9344 - loss: 0.2038
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9453 - loss: 0.1761
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9489 - loss: 0.1582
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.9512 - loss: 0.1504
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9534 - loss: 0.1452
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9574 - loss: 0.1303
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9608 - loss: 0.1214
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8729 - loss: 0.5101
```

Out[150…

```
[0.504758358001709, 0.8752400279045105]
```

loss is 0.50 and accuracy is 87.52%

hyper tuned parameters

In [152…

```
from tensorflow.keras import regularizers
np.random.seed(123)
model_Hyper = 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")
])
model_Hyper.compile(optimizer="rmsprop",
loss="mse",
metrics=["accuracy"])
np.random.seed(123)
history_model_Hyper = model_Hyper.fit(partial_a_train,partial_b_train,
epochs=20,
batch_size=512,
validation_data=(a_val, b_val))
history_dict_Hyper = history_model_Hyper.history
history_dict_Hyper.keys()
```

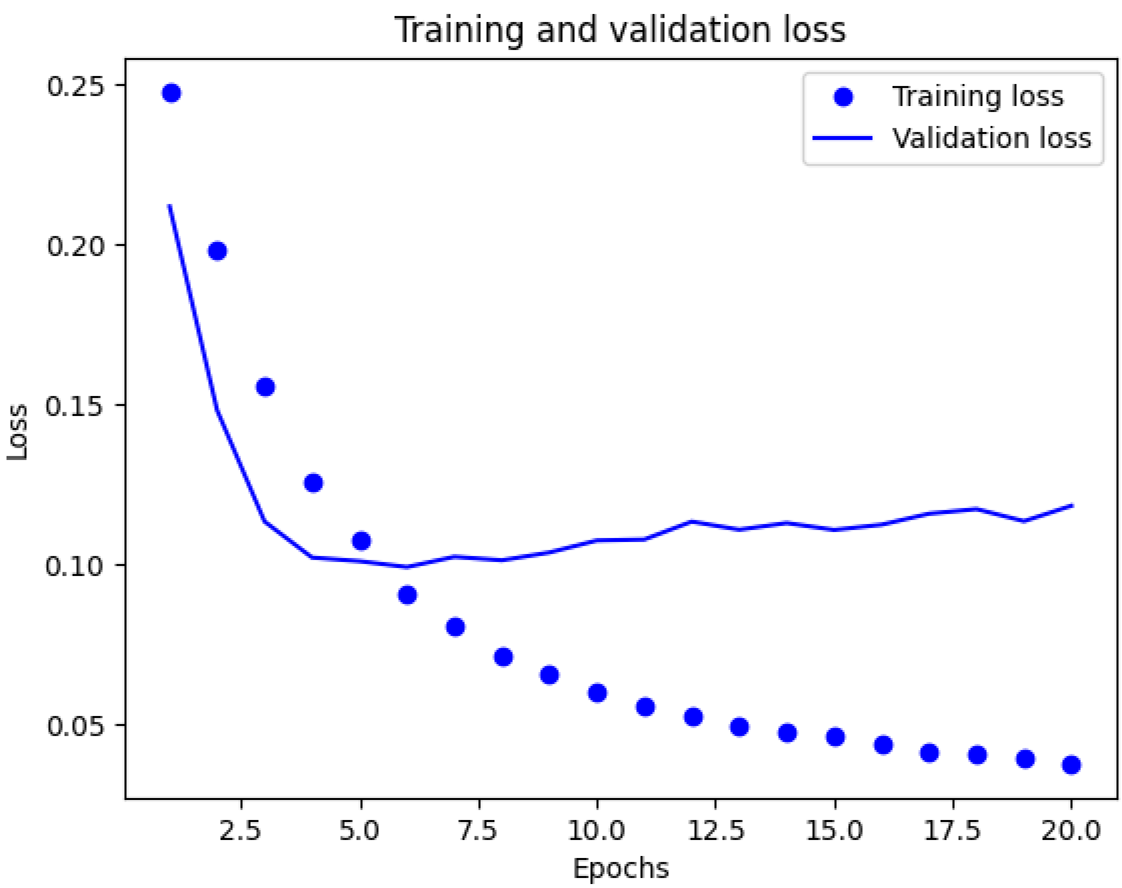
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 73ms/step - accuracy: 0.5494 - loss: 0.2557 - val_accu
racy: 0.8310 - val_loss: 0.2117
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.7196 - loss: 0.2090 - val_accu
racy: 0.8566 - val_loss: 0.1481
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.8160 - loss: 0.1623 - val_accu
racy: 0.8792 - val_loss: 0.1134
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.8637 - loss: 0.1289 - val_accu
racy: 0.8836 - val_loss: 0.1021
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.8938 - loss: 0.1084 - val_accu
racy: 0.8770 - val_loss: 0.1010
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - accuracy: 0.9109 - loss: 0.0924 - val_accu
racy: 0.8822 - val_loss: 0.0992
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - accuracy: 0.9295 - loss: 0.0798 - val_accu
racy: 0.8835 - val_loss: 0.1024
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9394 - loss: 0.0701 - val_accu
racy: 0.8866 - val_loss: 0.1013
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 68ms/step - accuracy: 0.9446 - loss: 0.0653 - val_accu
racy: 0.8856 - val_loss: 0.1037
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9506 - loss: 0.0598 - val_accu
racy: 0.8864 - val_loss: 0.1074
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9546 - loss: 0.0555 - val_accu
racy: 0.8849 - val_loss: 0.1078
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - accuracy: 0.9590 - loss: 0.0528 - val_accu
racy: 0.8776 - val_loss: 0.1133
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9632 - loss: 0.0492 - val_accu
racy: 0.8816 - val_loss: 0.1108
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 76ms/step - accuracy: 0.9644 - loss: 0.0474 - val_accu
racy: 0.8828 - val_loss: 0.1129
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.9652 - loss: 0.0464 - val_accu
racy: 0.8823 - val_loss: 0.1107
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9695 - loss: 0.0434 - val_accu
racy: 0.8825 - val_loss: 0.1124
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 0.9727 - loss: 0.0408 - val_accu
racy: 0.8799 - val_loss: 0.1158
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 105ms/step - accuracy: 0.9719 - loss: 0.0397 - val_acc
uracy: 0.8780 - val_loss: 0.1172
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - accuracy: 0.9718 - loss: 0.0398 - val_accu
```

```
racy: 0.8824 - val_loss: 0.1135
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - accuracy: 0.9751 - loss: 0.0370 - val_accu
racy: 0.8802 - val_loss: 0.1183
```

Out[152… dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])

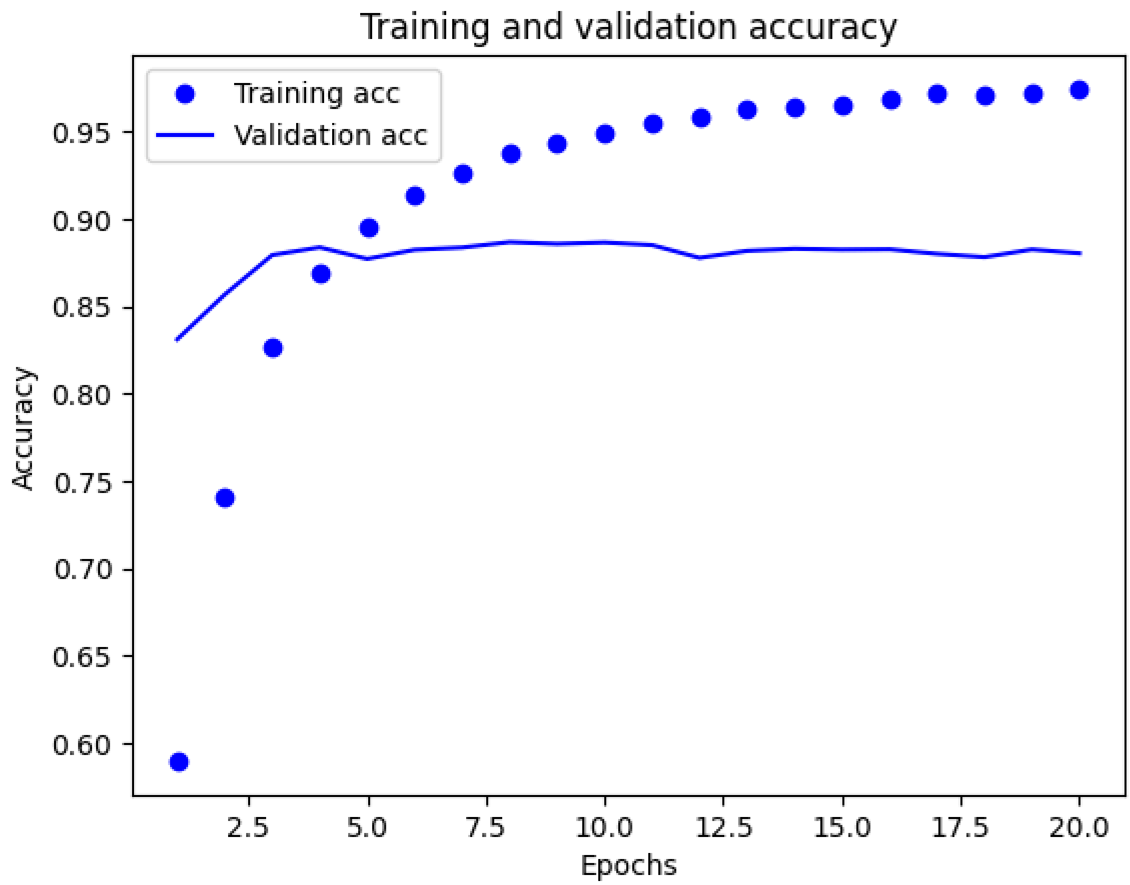
In [153…

```
loss_values = history_dict_Hyper["loss"]
val_loss_values = history_dict_Hyper["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()
```



In [154…

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



In [155…

```
model_Hyper.fit(a_train, b_train, epochs=8, batch_size=512)
results_Hyper = model_Hyper.evaluate(a_test, b_test)
results_Hyper
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.9295 - loss: 0.0744
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.9394 - loss: 0.0661
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9440 - loss: 0.0623
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9477 - loss: 0.0585
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.9531 - loss: 0.0548
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.9558 - loss: 0.0525
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 55ms/step - accuracy: 0.9607 - loss: 0.0491
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.9596 - loss: 0.0482
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8729 - loss: 0.1181
```

Out[155…

```
[0.11511095613241196, 0.8773199915885925]
```

In [161…

```
All_Models_Loss = np.array([results_Dropout[0], results_Hyper[0], results_MSE[0], r
All_Models_Accuracy = np.array([results_Dropout[1], results_Hyper[1], results_MSE[1


Labels = ['Model_Dropout', 'Model_Hyper', 'Model_MSE', 'Model_Regularization', 'Mod
```

```
plt.clf()
```

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

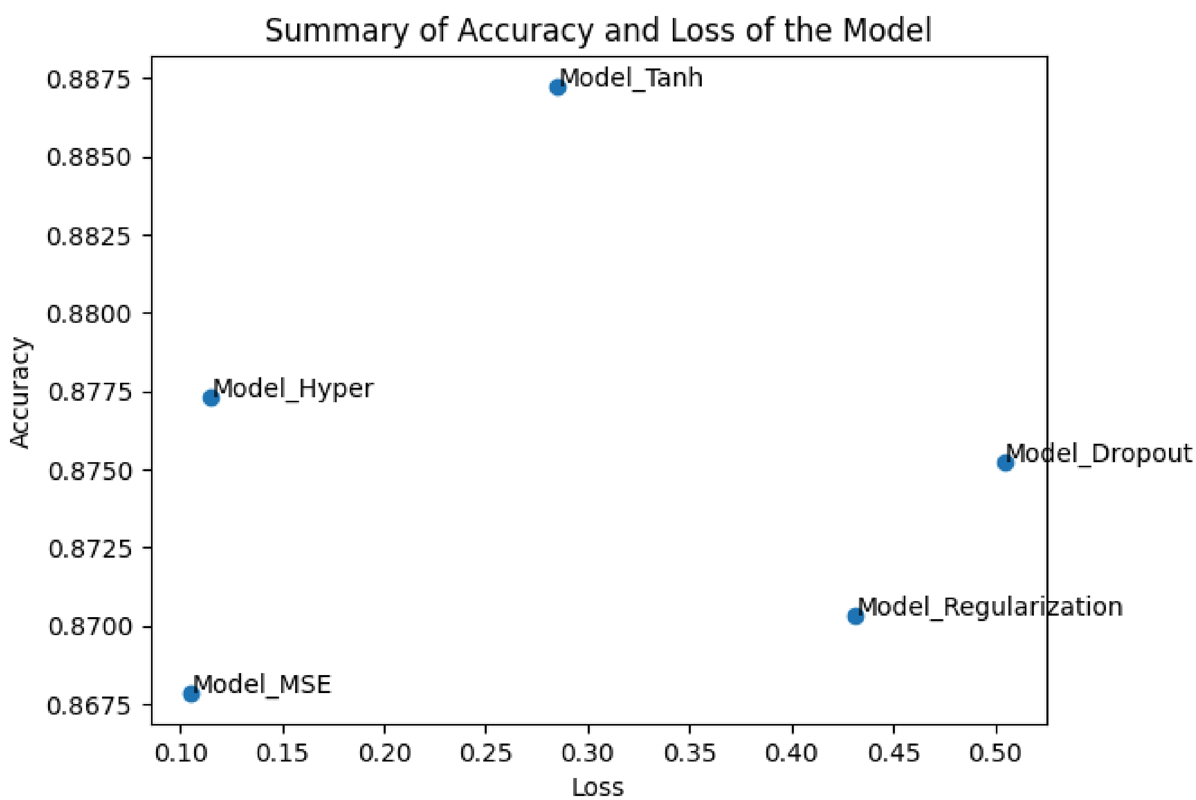
COMPILATION

In [163…

```
fig, ax = plt.subplots()
ax.scatter(All_Models_Loss, All_Models_Accuracy)

# Annotating each point with corresponding labels
for i, txt in enumerate(Labels):
    ax.annotate(txt, (All_Models_Loss[i], All_Models_Accuracy[i]))

plt.title("Summary of Accuracy and Loss of the Model")
plt.ylabel("Accuracy")
plt.xlabel("Loss")
plt.show()
```



SUMMARY OF RESULTS

In this study, various configurations of neural networks were tested on the IMDb dataset to differentiate between positive and negative reviews. The methods evaluated included varying the number of hidden layers, the number of nodes per layer, different activation and loss functions, and the application of the dropout technique.

Baseline Model Performance:

Architecture: The first model utilized ReLU activation, a binary cross-entropy loss function, and featured two hidden layers without applying any form of regularization. Accuracy: This baseline model achieved 88.41% accuracy on the validation set and 88.07% accuracy on the test set.

Effect of Increasing the Number of Hidden Layers:

Architecture: To improve the model's capacity to capture complex features, an additional hidden layer was added, bringing the total to three hidden layers. Accuracy: The model's performance slightly increased to 88% on the test set, indicating some improvement but also signs of overfitting with the added layers.

Regularization with Dropout

Architecture: Applying dropout to the layers helped prevent overfitting, enhancing the model's ability to generalize. Accuracy: With a dropout rate of 0.5, the model achieved 88.07% on the test set, demonstrating greater stability compared to models without regularization.

Activation Function Experimentation:

Sigmoid vs. ReLU: The sigmoid activation function was tested but performed worse compared to ReLU. Using ReLU resulted in better convergence with fewer iterations, achieving an accuracy of 88.78%, compared to 87.72% with Sigmoid.

Final Model Performance and Comparison

The best configuration was the model with three hidden layers, ReLU activation function, and dropout regularization, achieving a testing accuracy of 88.72%. This setup improved generalization and reduced overfitting compared to the baseline model. However, for the most complex architectures, learning did not accelerate as it initially did, highlighting the need to control the model’s complexity.

Conclusion: This experiment demonstrates that increasing model complexity can enable the learning of more intricate patterns. However, without regularization techniques, overfitting is likely to occur. For this task, employing dropout and an adequate number of layers emerged as the most effective strategies for improving accuracy.