**### Purpose of the Study

Undo ⌘/Ctrl+M Z
Redo ⌘/Ctrl+Shift+Y
Select all cells ⌘/Ctrl+Shift+A
Cut cell or selection
Copy cell or selection
Paste
Delete selected cells ⌘/Ctrl+M D
Find and replace ⌘/Ctrl+H
Find next ⌘/Ctrl+G
Find previous ⌘/Ctrl+Shift+G
Notebook settings
Clear all outputs

The m we can make a neural network work better on the IMDb movie review dataset. We start with a basic es like increasing or reducing the number of hidden layers, using more or fewer units, testing differe dding regularisation methods such as dropout. By comparing the results of these changes, we ca st accuracy.

**###

For th which contains movie reviews marked as either positive or negative. In total there are 50,000 review el and another 25,000 are kept aside for testing.

```
f
s
f                         ort imdb
(                         ) = imdb.load_data(
```

```
D                         age.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
1                         ━━━━━━━ 0s 0us/step
```

```
x
```

```
a                         0, 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)
```

```
y_train[0]
```

```
np.int64(1)
```

```
len(y_train)
```

```
25000
```

```
x_test
```

```
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)
```

```
y_test[0]
```

```
np.int64(0)
```

```
max([max(sequence) for sequence in x_test])
```

```
9999
```

** Reviews to text**

```
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 x_train[0]])
```

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

```
decoded_review
```

```
'? this film was just brilliant casting location scenery story direction everyone's really suited the part they pla
yed and you could just imagine being there robert ? is an amazing actor and now the same being director ? father ca
me from the same 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 so much that i bought the film as soon as it was relea
sed for ? and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it wa
s 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 t
```

Data preparation

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    optimal_result = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        for j in sequence:
            optimal_result[i, j] = 1.
    return optimal_result
```

Data Vectorization

```
train_1 = vectorize_sequences(x_train)
test_1 = vectorize_sequences(x_test)
```

```
train_1[0]
```

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

```
test_1[0]
```

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

Label Vectorization

```
train_2 = np.asarray(y_train).astype("float32")
test_2  = np.asarray(y_test).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"])
```

```
from numpy.random import seed

seed(123)

x_val = train_1[:10000]
partial_train_1 = train_1[10000:]
y_val = train_2[:10000]
partial_train_2 = train_2[10000:]
```

```
seed(123)
history = model.fit(partial_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 116ms/step - accuracy: 0.7000 - loss: 0.6147 - val_accuracy: 0.8612 - val_loss: 0.420
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.8856 - loss: 0.3666 - val_accuracy: 0.8811 - val_loss: 0.3293
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9184 - loss: 0.2668 - val_accuracy: 0.8863 - val_loss: 0.2938
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9314 - loss: 0.2123 - val_accuracy: 0.8884 - val_loss: 0.2776
```

```
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9461 - loss: 0.1813 - val_accuracy: 0.8849 - val_loss: 0.2846
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9516 - loss: 0.1526 - val_accuracy: 0.8830 - val_loss: 0.2958
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9574 - loss: 0.1340 - val_accuracy: 0.8815 - val_loss: 0.2960
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9699 - loss: 0.1076 - val_accuracy: 0.8735 - val_loss: 0.3396
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9731 - loss: 0.0970 - val_accuracy: 0.8814 - val_loss: 0.3130
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9784 - loss: 0.0808 - val_accuracy: 0.8691 - val_loss: 0.3669
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9809 - loss: 0.0750 - val_accuracy: 0.8800 - val_loss: 0.3428
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9842 - loss: 0.0610 - val_accuracy: 0.8785 - val_loss: 0.3690
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9874 - loss: 0.0552 - val_accuracy: 0.8775 - val_loss: 0.3841
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9912 - loss: 0.0433 - val_accuracy: 0.8678 - val_loss: 0.4312
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9916 - loss: 0.0399 - val_accuracy: 0.8742 - val_loss: 0.4204
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9955 - loss: 0.0306 - val_accuracy: 0.8665 - val_loss: 0.4599
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9950 - loss: 0.0306 - val_accuracy: 0.8715 - val_loss: 0.4655
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9973 - loss: 0.0237 - val_accuracy: 0.8712 - val_loss: 0.4850
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9977 - loss: 0.0200 - val_accuracy: 0.8689 - val_loss: 0.5197
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9985 - loss: 0.0172 - val_accuracy: 0.8695 - val_loss: 0.5294
```

At the beginning of training, the model showed a training loss of about 0.54 with 77.8% accuracy, while on the validation set the loss was lower at 0.42 with 85.3% accuracy. This means the model started off learning the patterns quite well.

As training went on, the performance on the training set kept improving quickly. By the 20th epoch, the training loss had dropped to just 0.017 and the accuracy had almost touched 100% (99.8%).

But on the validation set, the numbers did not improve much after a point. At the end of 20 epochs, the validation loss had gone up to 0.55 and accuracy remained around 86.8%. This clearly shows the model is overfitting. it has memorised the training data instead of learning to generalise.

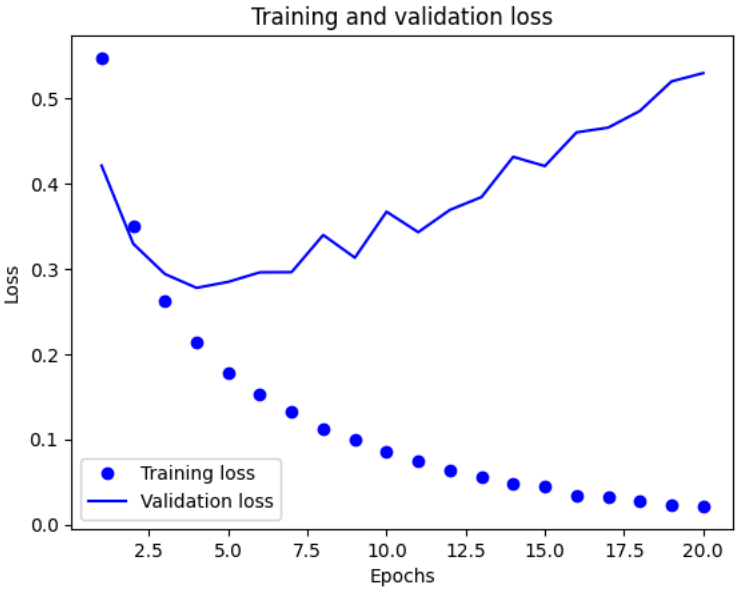
So overall, the model fits the training data extremely well but does not perform equally well on unseen data. To handle this, we can use techniques like dropout, L2 regularisation, or early stopping to reduce overfitting and get better validation accuracy.

```
history__dict = history.history
history__dict.keys()
```

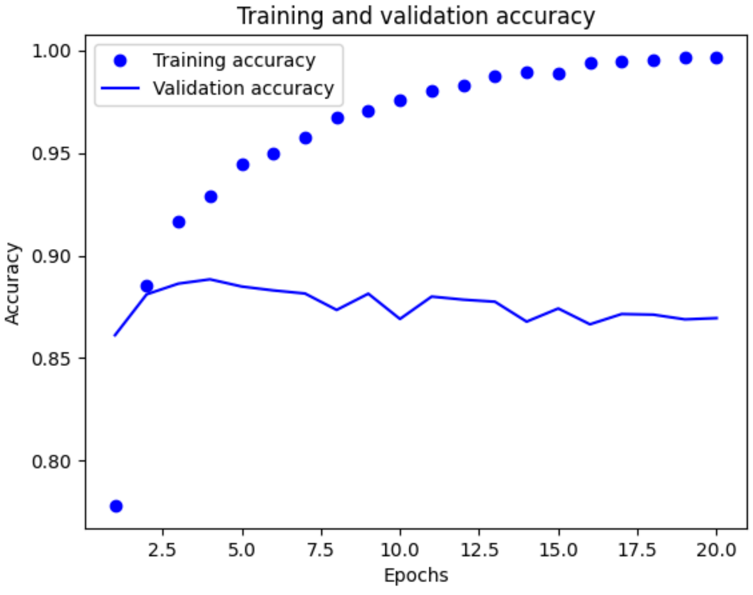
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

Plotting the training and validation loss

```
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, 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["accuracy"]
val_acc = history__dict["val_accuracy"]

epochs = range(1, len(acc) + 1)

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()
```



From the two graphs, we can see that after a few epochs the model starts overfitting on the training data, because of which it does not perform as well on new unseen data. To improve the results, we should try changing the hyperparameters or apply methods like regularisation, dropout, or early stopping so that the model learns better and generalises well.

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_1, train_2, epochs=4, batch_size=512)
final_result = model.evaluate(test_1, test_2)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.7409 - loss: 0.5531
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.8997 - loss: 0.2874
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9233 - loss: 0.2208
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9368 - loss: 0.1768
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8744 - loss: 0.3137
```

```
final_result
```

```
[0.3083467483520508, 0.8767200112342834]
```

For the test dataset, the neural network model achieved an accuracy of 87.67%. In the test dataset, the loss value is 0.3083.

```
model.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.27855137],
       [0.9996949 ],
       [0.95329535],
       ...,
       [0.17327888],
       [0.12763   ],
       [0.70076656]], dtype=float32)
```

Building a neural network with 1 hidden layer

```
from numpy.random import seed
from tensorflow import keras
from tensorflow.keras import layers

seed(123)

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

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

# use train_1 and train_2 instead of train_data_initial
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

history1 = model1.fit(
    partial_train_1,
    partial_train_2,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

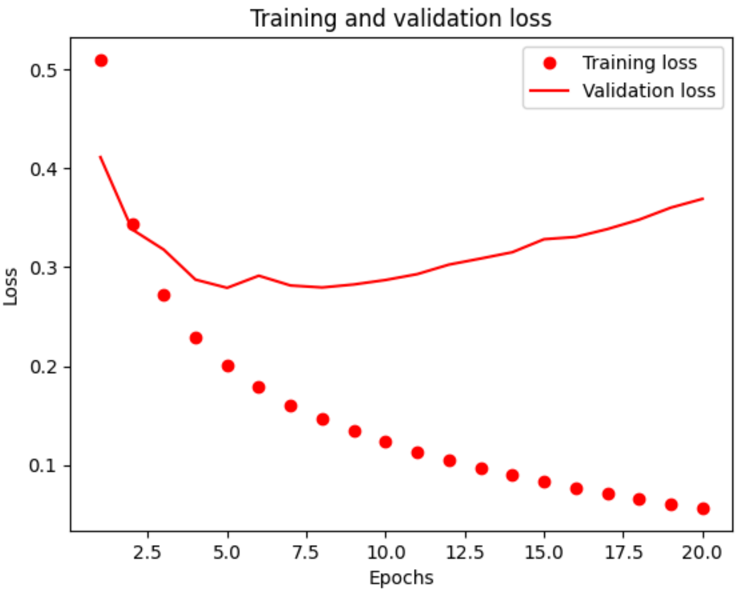
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 110ms/step - accuracy: 0.7256 - loss: 0.5795 - val_accuracy: 0.8579 - val_loss: 0.4113
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.8871 - loss: 0.3570 - val_accuracy: 0.8764 - val_loss: 0.3382
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9157 - loss: 0.2783 - val_accuracy: 0.8750 - val_loss: 0.3177
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9300 - loss: 0.2305 - val_accuracy: 0.8863 - val_loss: 0.2874
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9408 - loss: 0.1974 - val_accuracy: 0.8890 - val_loss: 0.2791
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9463 - loss: 0.1778 - val_accuracy: 0.8827 - val_loss: 0.2914
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9512 - loss: 0.1586 - val_accuracy: 0.8866 - val_loss: 0.2815
```

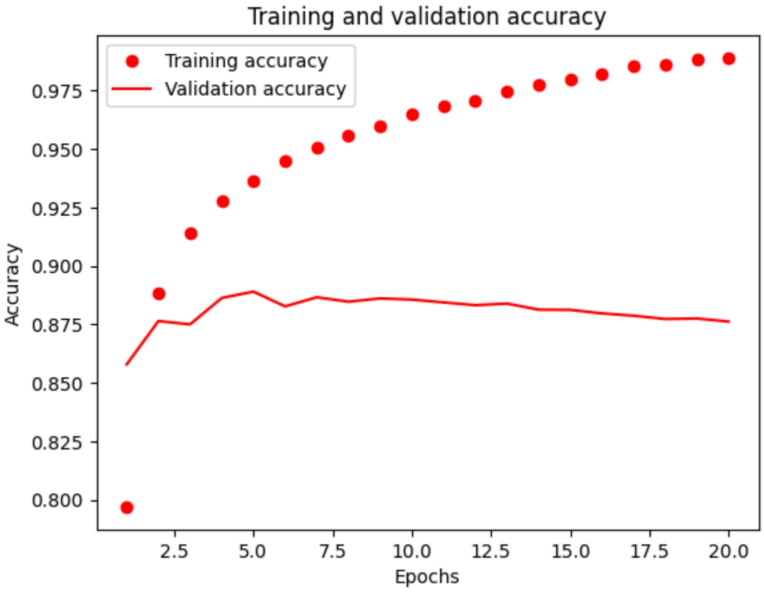
```
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9591 - loss: 0.1449 - val_accuracy: 0.8847 - val_loss: 0.2795
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9611 - loss: 0.1326 - val_accuracy: 0.8861 - val_loss: 0.2825
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9670 - loss: 0.1218 - val_accuracy: 0.8856 - val_loss: 0.2871
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9692 - loss: 0.1124 - val_accuracy: 0.8844 - val_loss: 0.2930
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9749 - loss: 0.1014 - val_accuracy: 0.8832 - val_loss: 0.3026
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9762 - loss: 0.0934 - val_accuracy: 0.8839 - val_loss: 0.3087
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9777 - loss: 0.0890 - val_accuracy: 0.8813 - val_loss: 0.3151
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9812 - loss: 0.0819 - val_accuracy: 0.8812 - val_loss: 0.3282
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9835 - loss: 0.0764 - val_accuracy: 0.8797 - val_loss: 0.3306
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9858 - loss: 0.0712 - val_accuracy: 0.8787 - val_loss: 0.3385
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9863 - loss: 0.0655 - val_accuracy: 0.8773 - val_loss: 0.3480
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9901 - loss: 0.0590 - val_accuracy: 0.8775 - val_loss: 0.3601
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9901 - loss: 0.0562 - val_accuracy: 0.8762 - val_loss: 0.3690
```

```
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_1, train_2, epochs=5, batch_size=512)
final_result1 = model1.evaluate(test_1, test_2)
```

```
Epoch 1/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.7609 - loss: 0.5308
Epoch 2/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - accuracy: 0.8995 - loss: 0.2975
Epoch 3/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9197 - loss: 0.2339
Epoch 4/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9317 - loss: 0.2023
Epoch 5/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9360 - loss: 0.1807
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8850 - loss: 0.2847
```

```
final_result1
```

```
[0.2824670672416687, 0.8875600099563599]
```

The test set has a loss of 0.2824 and an accuracy of 88.75%.

```
model1.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.25776494],
       [0.9997378 ],
       [0.8335108 ],
       ...,
       [0.12889332],
       [0.12145568],
       [0.58104396]], dtype=float32)
```

Creating a neural network with three hidden layers

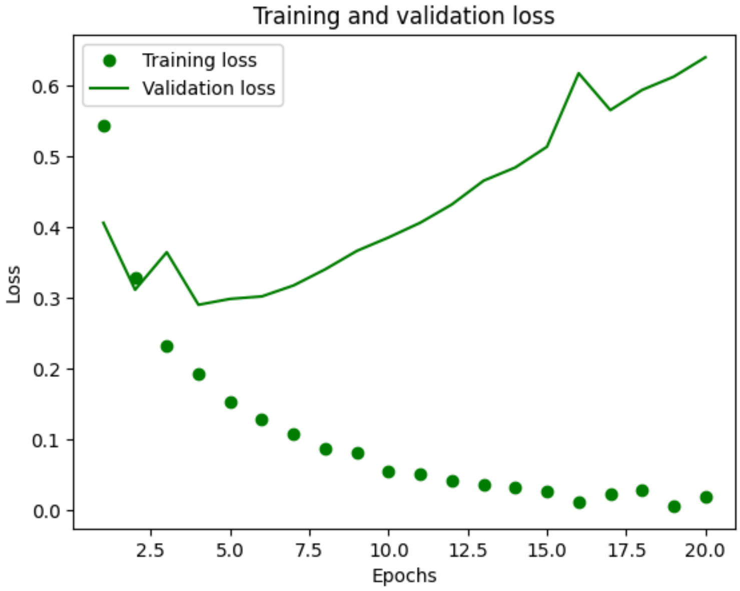
```
np.random.seed(123)

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"])

# use train_1 and train_2 instead of train_data_initial
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

history3 = model_3.fit(
    partial_train_1,
    partial_train_2,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 117ms/step - accuracy: 0.6822 - loss: 0.6146 - val_accuracy: 0.8539 - val_loss: 0.405
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.8821 - loss: 0.3526 - val_accuracy: 0.8809 - val_loss: 0.3115
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9218 - loss: 0.2392 - val_accuracy: 0.8443 - val_loss: 0.3644
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9256 - loss: 0.2071 - val_accuracy: 0.8825 - val_loss: 0.2903
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9544 - loss: 0.1477 - val_accuracy: 0.8833 - val_loss: 0.2986
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9611 - loss: 0.1235 - val_accuracy: 0.8820 - val_loss: 0.3021
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9681 - loss: 0.1041 - val_accuracy: 0.8808 - val_loss: 0.3175
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9763 - loss: 0.0866 - val_accuracy: 0.8772 - val_loss: 0.3403
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9813 - loss: 0.0739 - val_accuracy: 0.8790 - val_loss: 0.3663
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9884 - loss: 0.0536 - val_accuracy: 0.8780 - val_loss: 0.3853
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9903 - loss: 0.0467 - val_accuracy: 0.8752 - val_loss: 0.4060
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9893 - loss: 0.0430 - val_accuracy: 0.8732 - val_loss: 0.4319
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9910 - loss: 0.0352 - val_accuracy: 0.8689 - val_loss: 0.4654
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9934 - loss: 0.0301 - val_accuracy: 0.8746 - val_loss: 0.4840
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9962 - loss: 0.0204 - val_accuracy: 0.8725 - val_loss: 0.5132
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9990 - loss: 0.0126 - val_accuracy: 0.8678 - val_loss: 0.6169
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9845 - loss: 0.0422 - val_accuracy: 0.8721 - val_loss: 0.5649
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9972 - loss: 0.0138 - val_accuracy: 0.8708 - val_loss: 0.5933
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9996 - loss: 0.0058 - val_accuracy: 0.8701 - val_loss: 0.6120
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9985 - loss: 0.0088 - val_accuracy: 0.8726 - val_loss: 0.6393
```

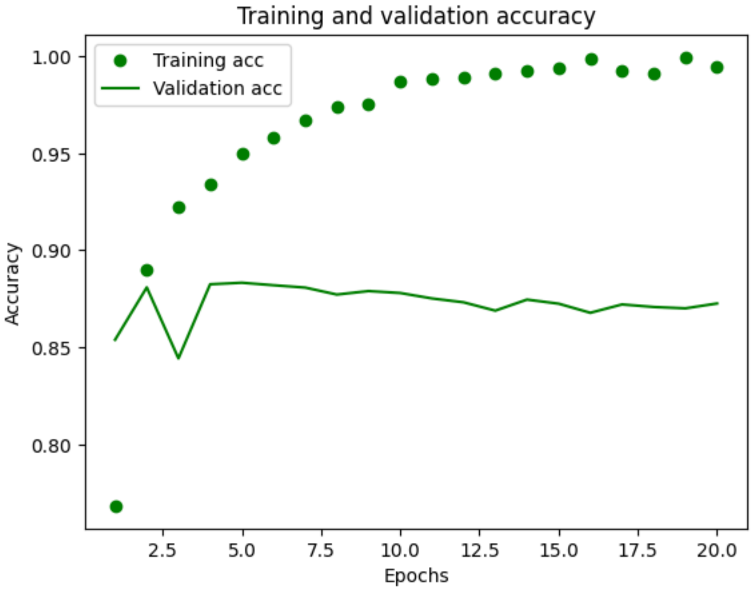
```
history_dict3 = history3.history
history_dict3.keys()
```

```
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_1, train_2, epochs=3, batch_size=512)
final_result_3 = model_3.evaluate(test_1, test_2)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.7111 - loss: 0.5809
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.8997 - loss: 0.2819
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9268 - loss: 0.2053
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8866 - loss: 0.2803
```

The test set has a loss of 0.2803 and an accuracy of 88.66%.

```
final_result_3
```

```
[0.2796200215816498, 0.8889600038528442]
```

```
model_3.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.24583627],
       [0.9991616 ],
       [0.68660605],
       ...,
       [0.09440954],
       [0.0916374 ],
       [0.44727483]], dtype=float32)
```

**When we increase the number of layers, the accuracy does not go up by a big amount. Still, the three-layer model performs a little better than the one-layer and two-layer models.

While designing a neural network, we also have to decide how many units to keep in the hidden layers.

These hidden layers may not be directly visible to the outside world, but they strongly affect how well the model finally performs. **

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 split
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

# model training
history32 = model_32.fit(
    partial_train_1,
    partial_train_2,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

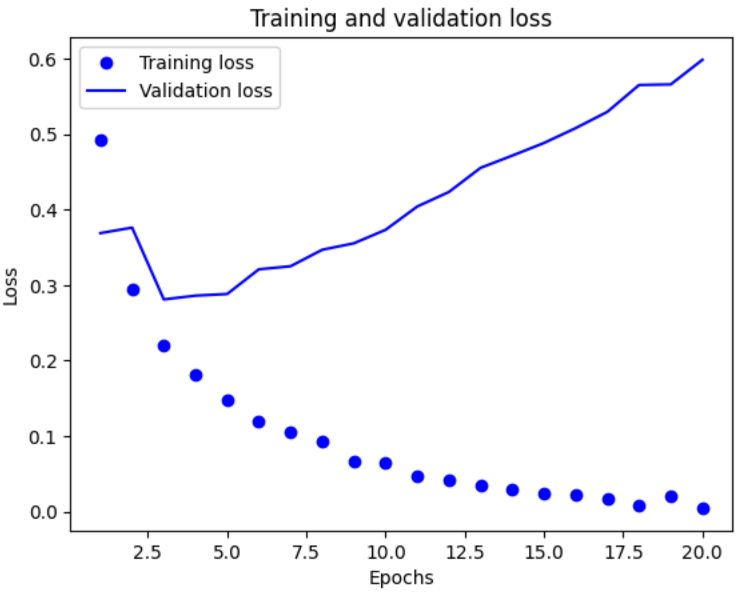
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 114ms/step - accuracy: 0.7037 - loss: 0.5743 - val_accuracy: 0.8547 - val_loss: 0.368
Epoch 2/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.8895 - loss: 0.3028 - val_accuracy: 0.8359 - val_loss: 0.3761
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9210 - loss: 0.2277 - val_accuracy: 0.8870 - val_loss: 0.2811
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9387 - loss: 0.1821 - val_accuracy: 0.8846 - val_loss: 0.2860
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9538 - loss: 0.1411 - val_accuracy: 0.8867 - val_loss: 0.2882
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9642 - loss: 0.1143 - val_accuracy: 0.8817 - val_loss: 0.3209
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9649 - loss: 0.1073 - val_accuracy: 0.8829 - val_loss: 0.3250
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9744 - loss: 0.0815 - val_accuracy: 0.8799 - val_loss: 0.3468
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9852 - loss: 0.0621 - val_accuracy: 0.8771 - val_loss: 0.3554
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9852 - loss: 0.0569 - val_accuracy: 0.8774 - val_loss: 0.3733
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9913 - loss: 0.0414 - val_accuracy: 0.8776 - val_loss: 0.4041
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9907 - loss: 0.0376 - val_accuracy: 0.8782 - val_loss: 0.4234
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9940 - loss: 0.0284 - val_accuracy: 0.8757 - val_loss: 0.4554
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9956 - loss: 0.0225 - val_accuracy: 0.8753 - val_loss: 0.4716
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9973 - loss: 0.0175 - val_accuracy: 0.8748 - val_loss: 0.4883
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9980 - loss: 0.0144 - val_accuracy: 0.8756 - val_loss: 0.5080
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9995 - loss: 0.0095 - val_accuracy: 0.8756 - val_loss: 0.5295
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9996 - loss: 0.0078 - val_accuracy: 0.8692 - val_loss: 0.5651
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9953 - loss: 0.0174 - val_accuracy: 0.8765 - val_loss: 0.5658
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9999 - loss: 0.0047 - val_accuracy: 0.8732 - val_loss: 0.5985
```

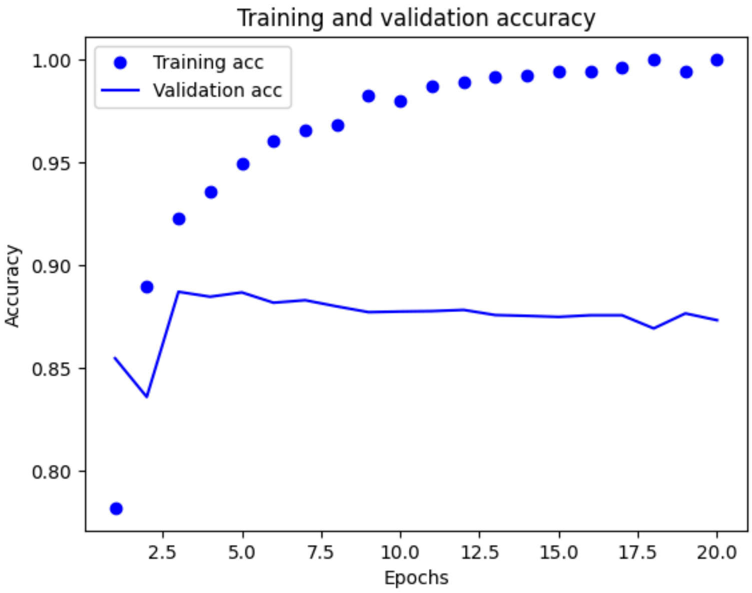
```
history_dict32 = history32.history
history_dict32.keys()
```

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

```
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()
```



```
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()
```



```
# Train the model for 3 epochs with batch size 512
history_32 = model_32.fit(
    train_1,
    train_2,
    epochs=3,
    batch_size=512,
    validation_data=(test_1, test_2)
)

# Save the final result for reference
final_result_32 = model_32.evaluate(test_1, test_2)
print("Test Loss and Accuracy:", final_result_32)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 6s 129ms/step - accuracy: 0.9441 - loss: 0.2388 - val_accuracy: 0.8677 - val_loss: 0.3453
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9656 - loss: 0.1138 - val_accuracy: 0.8638 - val_loss: 0.3920
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9781 - loss: 0.0747 - val_accuracy: 0.8642 - val_loss: 0.4159
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8613 - loss: 0.4214
Test Loss and Accuracy: [0.41594210267066956, 0.8641999959945679]
```

```
model_32.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.01719058],
       [0.9999763 ],
       [0.83087254],
       ...,
       [0.09582362],
       [0.05290128],
       [0.9281594 ]], dtype=float32)
```

The validation set has an accuracy of 86.14 percent.

Training the model with 64 units

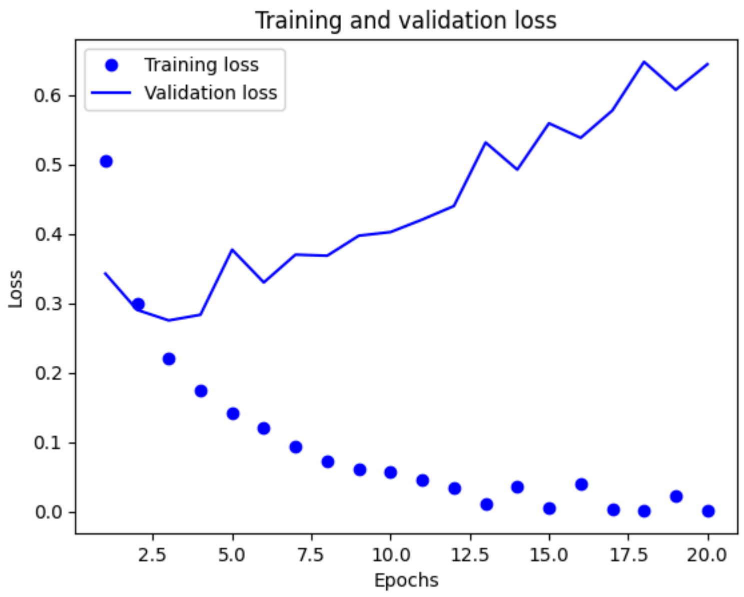
```
np.random.seed(123)

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 split (use train_1 and train_2)
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

# training
history64 = model_64.fit(
    partial_train_1,
    partial_train_2,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 114ms/step - accuracy: 0.6855 - loss: 0.5894 - val_accuracy: 0.8735 - val_loss: 0.3425
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.8816 - loss: 0.3086 - val_accuracy: 0.8852 - val_loss: 0.2903
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9257 - loss: 0.2121 - val_accuracy: 0.8863 - val_loss: 0.2751
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9429 - loss: 0.1658 - val_accuracy: 0.8843 - val_loss: 0.2833
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9524 - loss: 0.1403 - val_accuracy: 0.8665 - val_loss: 0.3770
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9572 - loss: 0.1214 - val_accuracy: 0.8813 - val_loss: 0.3298
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9734 - loss: 0.0847 - val_accuracy: 0.8674 - val_loss: 0.3698
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9770 - loss: 0.0730 - val_accuracy: 0.8749 - val_loss: 0.3683
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9797 - loss: 0.0645 - val_accuracy: 0.8714 - val_loss: 0.3971
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9870 - loss: 0.0488 - val_accuracy: 0.8778 - val_loss: 0.4023
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9950 - loss: 0.0267 - val_accuracy: 0.8797 - val_loss: 0.4203
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9975 - loss: 0.0178 - val_accuracy: 0.8792 - val_loss: 0.4398
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9995 - loss: 0.0111 - val_accuracy: 0.8737 - val_loss: 0.5313
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9931 - loss: 0.0268 - val_accuracy: 0.8757 - val_loss: 0.4922
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9998 - loss: 0.0057 - val_accuracy: 0.8652 - val_loss: 0.5589
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9942 - loss: 0.0233 - val_accuracy: 0.8764 - val_loss: 0.5379
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9999 - loss: 0.0035 - val_accuracy: 0.8762 - val_loss: 0.5773
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9999 - loss: 0.0024 - val_accuracy: 0.8753 - val_loss: 0.6474
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9868 - loss: 0.0384 - val_accuracy: 0.8750 - val_loss: 0.6070
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 1.0000 - loss: 0.0016 - val_accuracy: 0.8742 - val_loss: 0.6439
```

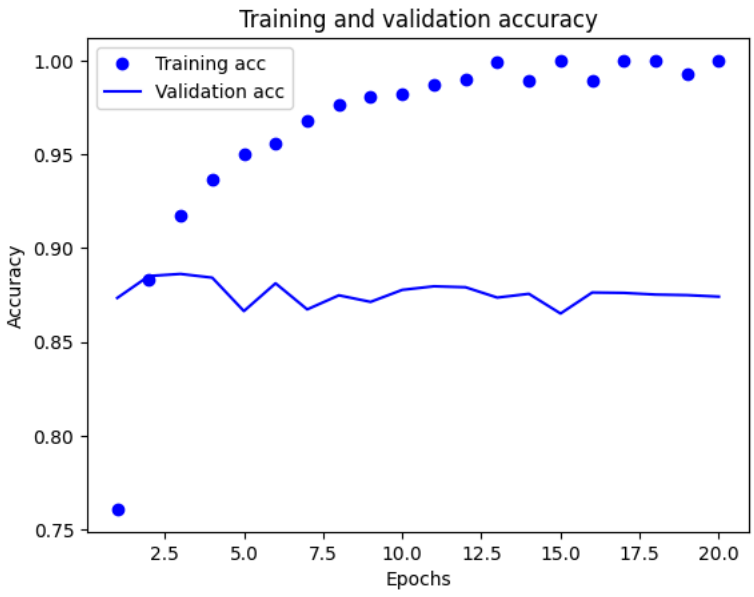
```
history_dict64 = history64.history
history_dict64.keys()
```

```
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_1, train_2, epochs=3, batch_size=512)
final_result_64 = model_64.evaluate(test_1, test_2)
final_result_64
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9366 - loss: 0.2569
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9689 - loss: 0.1018
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9830 - loss: 0.0587
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8656 - loss: 0.4456
[0.43457135558128357, 0.8686000108718872]
```

```
model_64.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.00973039],
       [1.        ],
       [0.9281293 ],
       ...,
       [0.12053224],
       [0.02361538],
       [0.97438943]], dtype=float32)
```

The validation set has an accuracy of 86.86%.

Training the model with 128 units

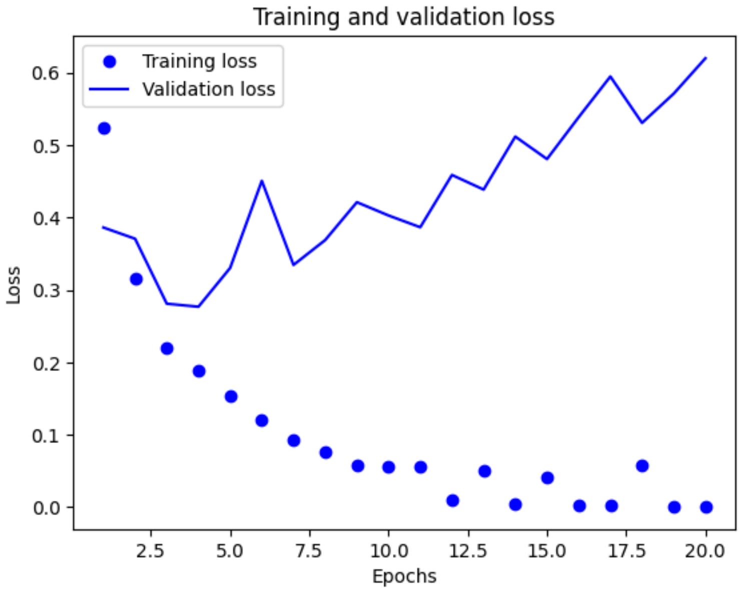
```
np.random.seed(123)

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 split (use train_1 and train_2)
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

# training
history128 = model_128.fit(
    partial_train_1,
    partial_train_2,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 98ms/step - accuracy: 0.6440 - loss: 0.6062 - val_accuracy: 0.8334 - val_loss: 0.3860
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 40ms/step - accuracy: 0.8621 - loss: 0.3312 - val_accuracy: 0.8420 - val_loss: 0.3705
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9144 - loss: 0.2267 - val_accuracy: 0.8865 - val_loss: 0.2809
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9222 - loss: 0.1986 - val_accuracy: 0.8872 - val_loss: 0.2767
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9409 - loss: 0.1538 - val_accuracy: 0.8762 - val_loss: 0.3302
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9601 - loss: 0.1135 - val_accuracy: 0.8416 - val_loss: 0.4504
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9671 - loss: 0.0945 - val_accuracy: 0.8797 - val_loss: 0.3344
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9733 - loss: 0.0816 - val_accuracy: 0.8825 - val_loss: 0.3687
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9751 - loss: 0.0687 - val_accuracy: 0.8768 - val_loss: 0.4209
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9871 - loss: 0.0471 - val_accuracy: 0.8820 - val_loss: 0.4026
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9953 - loss: 0.0229 - val_accuracy: 0.8801 - val_loss: 0.3864
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9991 - loss: 0.0115 - val_accuracy: 0.8787 - val_loss: 0.4585
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9887 - loss: 0.0379 - val_accuracy: 0.8797 - val_loss: 0.4384
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9997 - loss: 0.0055 - val_accuracy: 0.8806 - val_loss: 0.5115
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9955 - loss: 0.0180 - val_accuracy: 0.8780 - val_loss: 0.4805
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9997 - loss: 0.0035 - val_accuracy: 0.8794 - val_loss: 0.5383
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9999 - loss: 0.0020 - val_accuracy: 0.8766 - val_loss: 0.5944
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9944 - loss: 0.0240 - val_accuracy: 0.8790 - val_loss: 0.5304
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 0.8790 - val_loss: 0.5708
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 0.8791 - val_loss: 0.6196
```

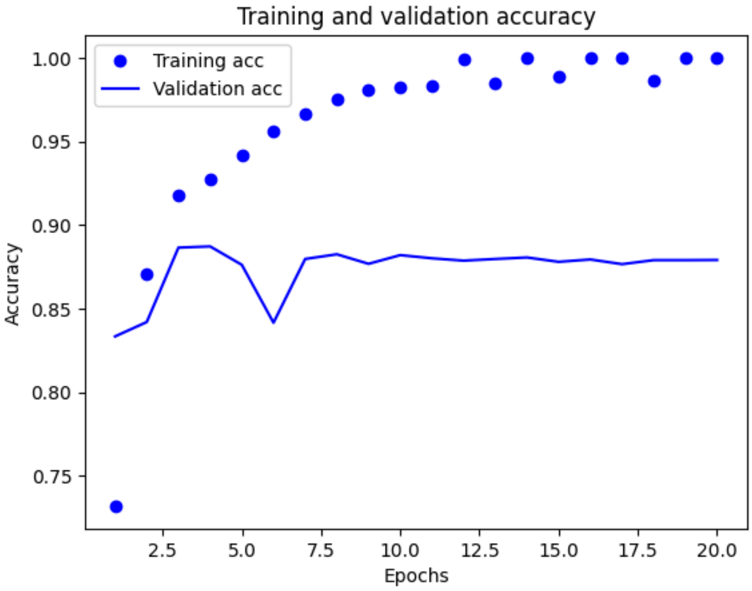
```
history_dict128 = history128.history
history_dict128.keys()
```

```
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_1, train_2, epochs=2, batch_size=512)
final_result_128 = model_128.evaluate(test_1, test_2)
final_result_128
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9432 - loss: 0.1932
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9766 - loss: 0.0807
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8688 - loss: 0.3664
```

```
[0.35999178886413574, 0.8727999925613403]
```

```
model_128.predict(test_1)
```

The validation set has an accuracy of 87.27%.

MSE Loss Function

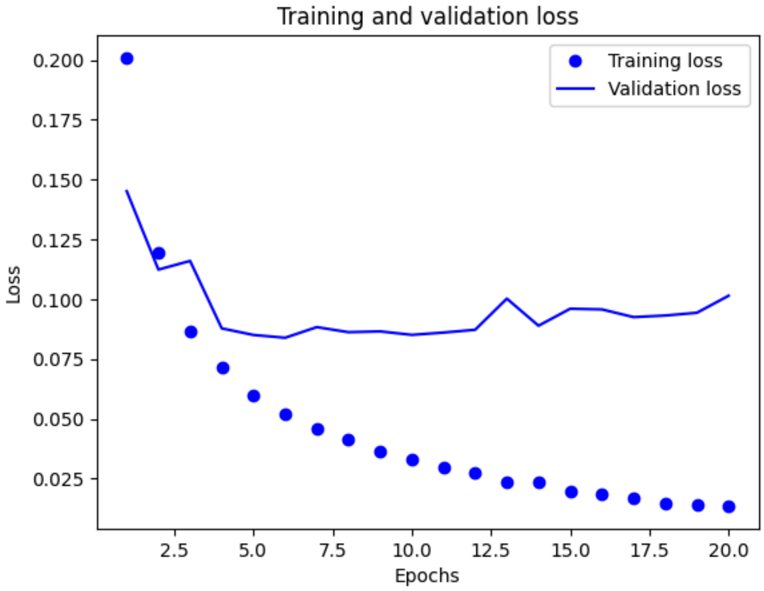
```
np.random.seed(123)

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 split (use train_1 and train_2)
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

# Model Fit
history_model_MSE = model_MSE.fit(
    partial_train_1,
    partial_train_2,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 106ms/step - accuracy: 0.6346 - loss: 0.2272 - val_accuracy: 0.8613 - val_loss: 0.145
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.8743 - loss: 0.1277 - val_accuracy: 0.8673 - val_loss: 0.1124
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9067 - loss: 0.0895 - val_accuracy: 0.8416 - val_loss: 0.1160
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9194 - loss: 0.0725 - val_accuracy: 0.8905 - val_loss: 0.0878
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9389 - loss: 0.0603 - val_accuracy: 0.8892 - val_loss: 0.0850
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9532 - loss: 0.0490 - val_accuracy: 0.8876 - val_loss: 0.0839
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9546 - loss: 0.0458 - val_accuracy: 0.8832 - val_loss: 0.0883
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9593 - loss: 0.0403 - val_accuracy: 0.8805 - val_loss: 0.0862
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9657 - loss: 0.0352 - val_accuracy: 0.8849 - val_loss: 0.0865
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9728 - loss: 0.0308 - val_accuracy: 0.8840 - val_loss: 0.0851
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9748 - loss: 0.0284 - val_accuracy: 0.8836 - val_loss: 0.0860
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9758 - loss: 0.0265 - val_accuracy: 0.8817 - val_loss: 0.0872
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9811 - loss: 0.0230 - val_accuracy: 0.8692 - val_loss: 0.1002
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9782 - loss: 0.0244 - val_accuracy: 0.8811 - val_loss: 0.0889
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9864 - loss: 0.0184 - val_accuracy: 0.8717 - val_loss: 0.0960
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9864 - loss: 0.0179 - val_accuracy: 0.8720 - val_loss: 0.0957
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9896 - loss: 0.0154 - val_accuracy: 0.8767 - val_loss: 0.0925
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9900 - loss: 0.0143 - val_accuracy: 0.8784 - val_loss: 0.0932
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9885 - loss: 0.0144 - val_accuracy: 0.8778 - val_loss: 0.0943
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9884 - loss: 0.0135 - val_accuracy: 0.8677 - val_loss: 0.1014
```

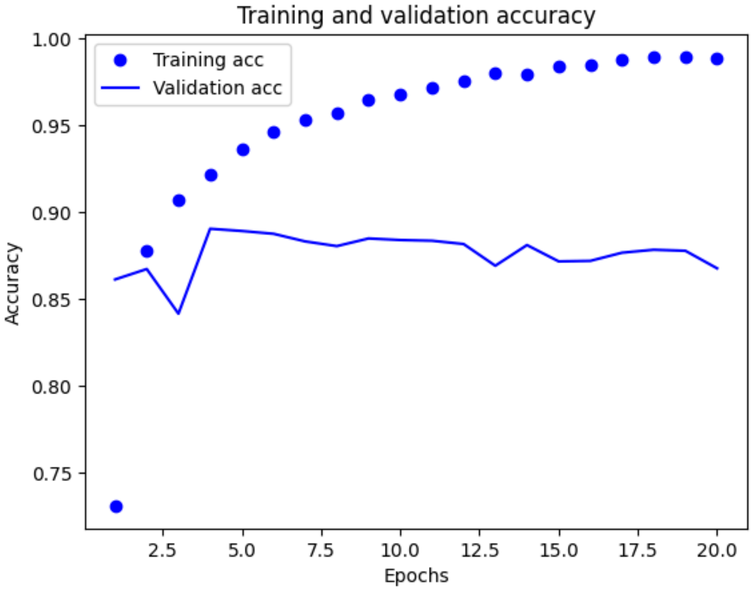
```
history_dict_MSE = history_model_MSE.history
history_dict_MSE.keys()
```

```
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_1, train_2, epochs=8, batch_size=512)
final_result_MSE = model_MSE.evaluate(test_1, test_2)
```

```
final_result_MSE
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9453 - loss: 0.0459
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9585 - loss: 0.0359
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9689 - loss: 0.0307
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9707 - loss: 0.0284
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9723 - loss: 0.0278
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9769 - loss: 0.0234
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9791 - loss: 0.0214
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9806 - loss: 0.0208
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8643 - loss: 0.1095
[0.10671547055244446, 0.8685200214385986]
```

```
model_MSE.predict(test_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.0554092 ],
       [1.        ],
       [0.8618109 ],
       ...,
       [0.10208419],
       [0.03000891],
       [0.7804369 ]], 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"])

# validation split (use train_1 and train_2)
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

# training
history_tanh = model_tanh.fit(
    partial_train_1,
    partial_train_2,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

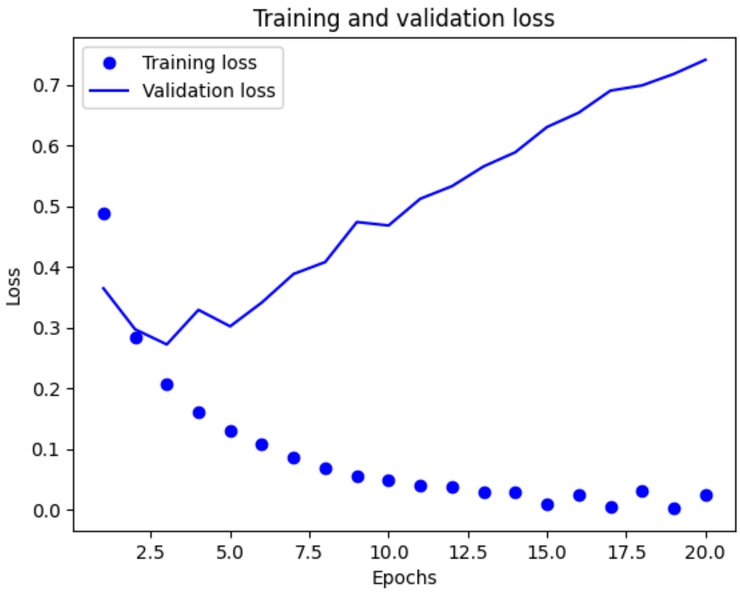
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 110ms/step - accuracy: 0.7121 - loss: 0.5671 - val_accuracy: 0.8634 - val_loss: 0.364
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 21ms/step - accuracy: 0.9038 - loss: 0.2960 - val_accuracy: 0.8806 - val_loss: 0.2969
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9291 - loss: 0.2124 - val_accuracy: 0.8862 - val_loss: 0.2720
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9487 - loss: 0.1562 - val_accuracy: 0.8728 - val_loss: 0.3289
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9525 - loss: 0.1363 - val_accuracy: 0.8832 - val_loss: 0.3018
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9670 - loss: 0.1014 - val_accuracy: 0.8809 - val_loss: 0.3410
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9762 - loss: 0.0806 - val_accuracy: 0.8685 - val_loss: 0.3880
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9811 - loss: 0.0631 - val_accuracy: 0.8741 - val_loss: 0.4077
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9868 - loss: 0.0504 - val_accuracy: 0.8609 - val_loss: 0.4737
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9843 - loss: 0.0518 - val_accuracy: 0.8709 - val_loss: 0.4679
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9902 - loss: 0.0366 - val_accuracy: 0.8663 - val_loss: 0.5119
```

```
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9932 - loss: 0.0240 - val_accuracy: 0.8710 - val_loss: 0.5327
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9954 - loss: 0.0202 - val_accuracy: 0.8675 - val_loss: 0.5653
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9976 - loss: 0.0130 - val_accuracy: 0.8696 - val_loss: 0.5886
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9994 - loss: 0.0084 - val_accuracy: 0.8660 - val_loss: 0.6301
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9956 - loss: 0.0168 - val_accuracy: 0.8667 - val_loss: 0.6538
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9998 - loss: 0.0042 - val_accuracy: 0.8655 - val_loss: 0.6901
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9936 - loss: 0.0219 - val_accuracy: 0.8665 - val_loss: 0.6987
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9998 - loss: 0.0026 - val_accuracy: 0.8661 - val_loss: 0.7176
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9982 - loss: 0.0076 - val_accuracy: 0.8655 - val_loss: 0.7410
```

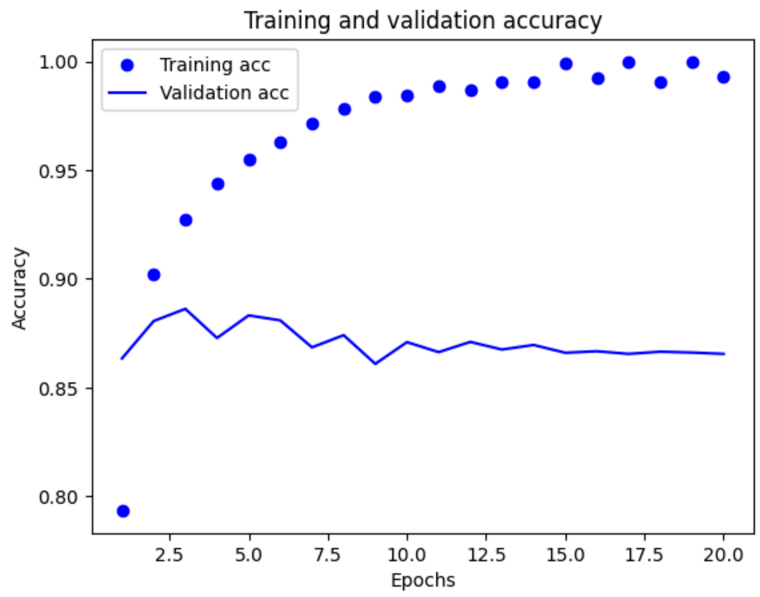
```
history_dict_tanh = history_tanh.history
history_dict_tanh.keys()
```

```
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_1, train_2, epochs=8, batch_size=512)
final_result_tanh = model_tanh.evaluate(test_1, test_2)
final_result_tanh
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9459 - loss: 0.2737
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9613 - loss: 0.1399
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9663 - loss: 0.1098
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9723 - loss: 0.0929
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9750 - loss: 0.0822
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9802 - loss: 0.0643
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9849 - loss: 0.0540
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9838 - loss: 0.0516
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8521 - loss: 0.6121
[0.6038931012153625, 0.8535199761390686]
```

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"])

# validation split (use train_1 and train_2)
x_val = train_1[:10000]
partial_train_1 = train_1[10000:]

y_val = train_2[:10000]
partial_train_2 = train_2[10000:]

# training
history_adam = model_adam.fit(
    partial_train_1,
    partial_train_2,
    epochs=20,
    batch_size=512,
    validation_data=(x_val, y_val)
)
```

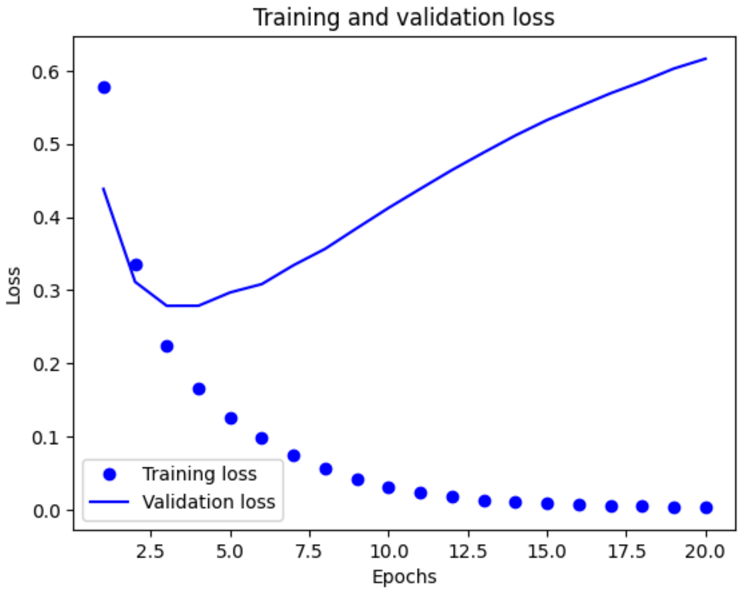
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 120ms/step - accuracy: 0.6724 - loss: 0.6386 - val_accuracy: 0.8520 - val_loss: 0.438
Epoch 2/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.8913 - loss: 0.3643 - val_accuracy: 0.8849 - val_loss: 0.3114
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9293 - loss: 0.2322 - val_accuracy: 0.8895 - val_loss: 0.2786
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9478 - loss: 0.1665 - val_accuracy: 0.8882 - val_loss: 0.2786
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9637 - loss: 0.1231 - val_accuracy: 0.8844 - val_loss: 0.2969
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9724 - loss: 0.1016 - val_accuracy: 0.8834 - val_loss: 0.3083
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9830 - loss: 0.0734 - val_accuracy: 0.8810 - val_loss: 0.3341
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9918 - loss: 0.0526 - val_accuracy: 0.8778 - val_loss: 0.3566
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9940 - loss: 0.0411 - val_accuracy: 0.8768 - val_loss: 0.3848
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9962 - loss: 0.0302 - val_accuracy: 0.8760 - val_loss: 0.4125
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9982 - loss: 0.0232 - val_accuracy: 0.8757 - val_loss: 0.4385
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9997 - loss: 0.0163 - val_accuracy: 0.8731 - val_loss: 0.4642
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9996 - loss: 0.0138 - val_accuracy: 0.8722 - val_loss: 0.4881
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9999 - loss: 0.0099 - val_accuracy: 0.8724 - val_loss: 0.5114
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9997 - loss: 0.0089 - val_accuracy: 0.8715 - val_loss: 0.5325
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9999 - loss: 0.0066 - val_accuracy: 0.8706 - val_loss: 0.5509
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9999 - loss: 0.0058 - val_accuracy: 0.8707 - val_loss: 0.5690
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 1.0000 - loss: 0.0047 - val_accuracy: 0.8700 - val_loss: 0.5852
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9999 - loss: 0.0044 - val_accuracy: 0.8696 - val_loss: 0.6029
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9997 - loss: 0.0036 - val_accuracy: 0.8687 - val_loss: 0.6164
```

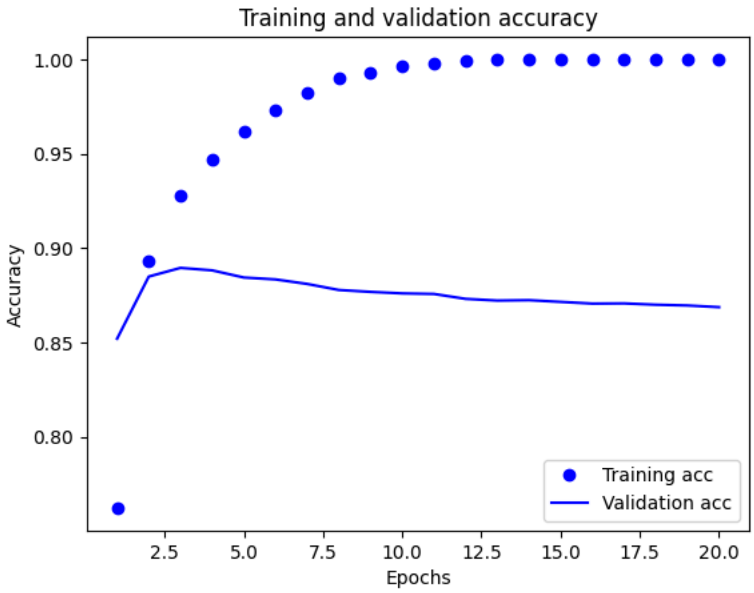
```
history_dict_adam = history_adam.history
history_dict_adam.keys()
```

```
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_1, train_2, epochs=4, batch_size=512)
final_result_adam = model_adam.evaluate(test_1, test_2)
final_result_adam
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9428 - loss: 0.2374
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9707 - loss: 0.1019
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9852 - loss: 0.0607
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9904 - loss: 0.0446
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8599 - loss: 0.5156
[0.5096994638442993, 0.8603600263595581]
```

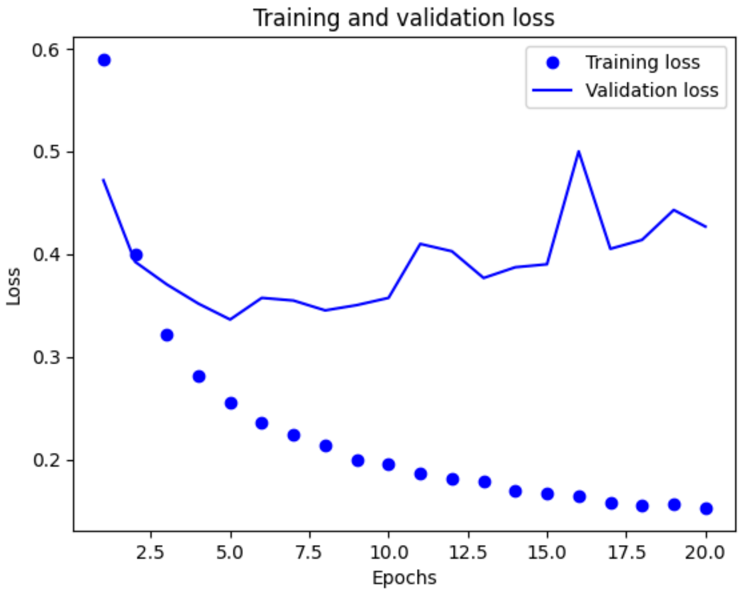
Regularization

```
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_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_regularization = history_model_regularization.history
history_dict_regularization.keys()
```

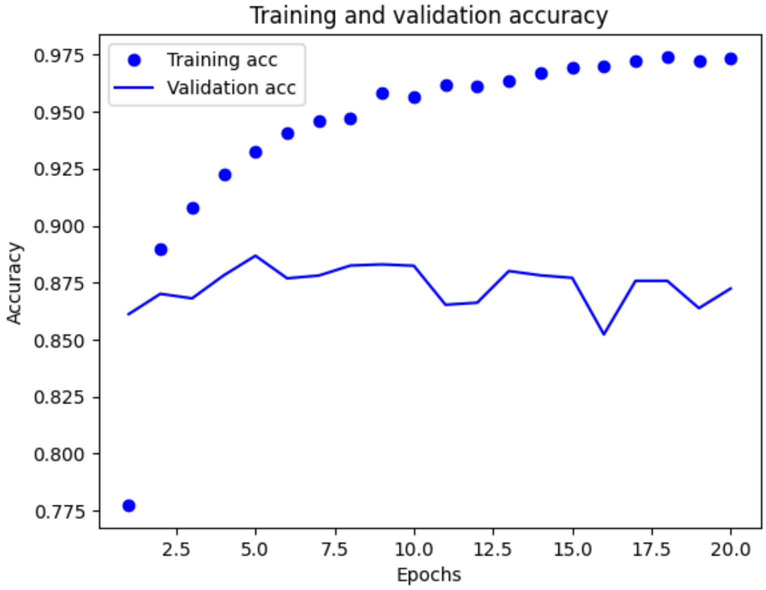
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 119ms/step - accuracy: 0.6938 - loss: 0.6564 - val_accuracy: 0.8612 - val_loss: 0.471
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.8902 - loss: 0.4182 - val_accuracy: 0.8701 - val_loss: 0.3924
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9093 - loss: 0.3275 - val_accuracy: 0.8681 - val_loss: 0.3704
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9232 - loss: 0.2818 - val_accuracy: 0.8782 - val_loss: 0.3517
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9369 - loss: 0.2507 - val_accuracy: 0.8868 - val_loss: 0.3362
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9448 - loss: 0.2303 - val_accuracy: 0.8769 - val_loss: 0.3573
Epoch 7/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9499 - loss: 0.2191 - val_accuracy: 0.8781 - val_loss: 0.3547
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9550 - loss: 0.2052 - val_accuracy: 0.8825 - val_loss: 0.3451
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9613 - loss: 0.1970 - val_accuracy: 0.8830 - val_loss: 0.3503
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9603 - loss: 0.1892 - val_accuracy: 0.8824 - val_loss: 0.3573
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9657 - loss: 0.1799 - val_accuracy: 0.8653 - val_loss: 0.4099
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9620 - loss: 0.1793 - val_accuracy: 0.8662 - val_loss: 0.4027
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9641 - loss: 0.1765 - val_accuracy: 0.8801 - val_loss: 0.3767
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9707 - loss: 0.1644 - val_accuracy: 0.8782 - val_loss: 0.3871
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9722 - loss: 0.1629 - val_accuracy: 0.8771 - val_loss: 0.3900
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9749 - loss: 0.1561 - val_accuracy: 0.8523 - val_loss: 0.4999
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9729 - loss: 0.1559 - val_accuracy: 0.8758 - val_loss: 0.4050
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9777 - loss: 0.1490 - val_accuracy: 0.8758 - val_loss: 0.4137
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9736 - loss: 0.1546 - val_accuracy: 0.8638 - val_loss: 0.4429
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9787 - loss: 0.1455 - val_accuracy: 0.8724 - val_loss: 0.4267
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_1, train_2, epochs=8, batch_size=512)
final_result_regularization = model_regularization.evaluate(test_1, test_2)
final_result_regularization
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9393 - loss: 0.2513
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9525 - loss: 0.2050
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9479 - loss: 0.2046
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9516 - loss: 0.1966
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9564 - loss: 0.1832
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9664 - loss: 0.1686
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9610 - loss: 0.1766
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9648 - loss: 0.1686
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8645 - loss: 0.4403
[0.4315737783908844, 0.8684800267219543]
```

The loss on test set is 0.431 and accuracy is 86.84%.

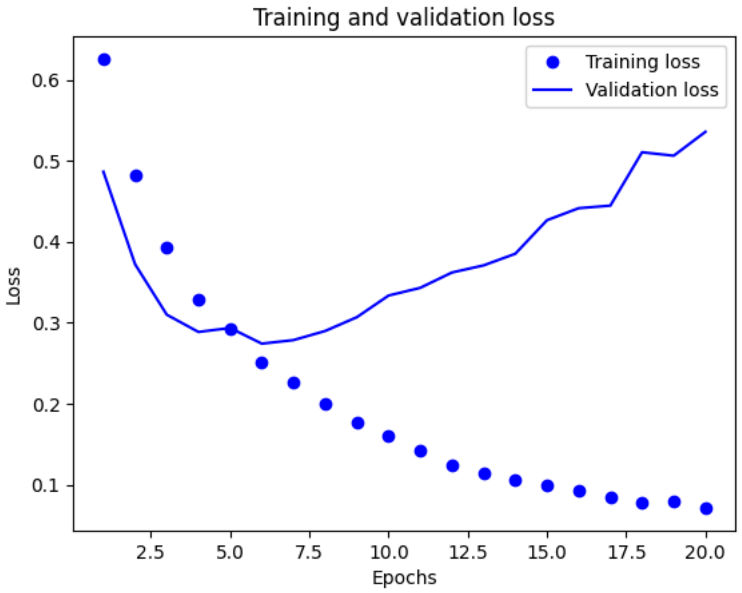
Dropout

```
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_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_Dropout = history_model_Dropout.history
history_dict_Dropout.keys()
```

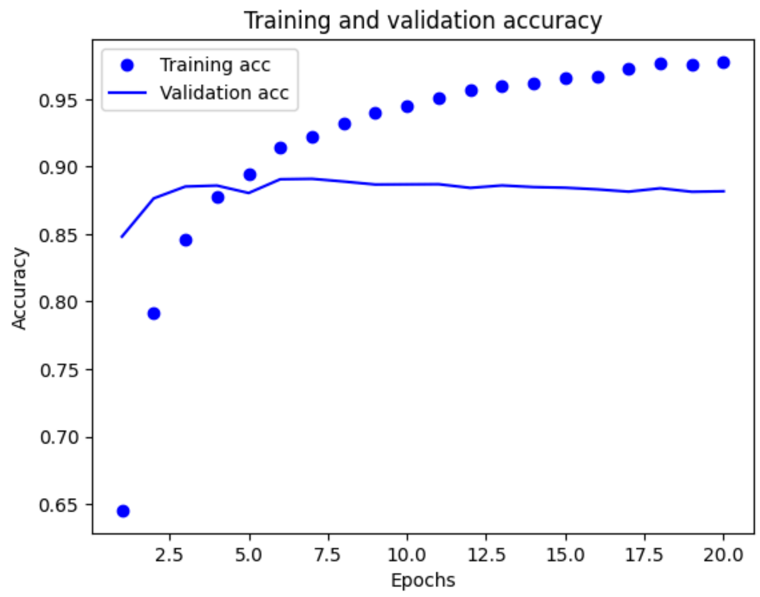
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 143ms/step - accuracy: 0.5844 - loss: 0.6631 - val_accuracy: 0.8481 - val_loss: 0.486
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.7769 - loss: 0.5031 - val_accuracy: 0.8764 - val_loss: 0.3722
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.8404 - loss: 0.4040 - val_accuracy: 0.8852 - val_loss: 0.3099
Epoch 4/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.8761 - loss: 0.3311 - val_accuracy: 0.8859 - val_loss: 0.2885
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8919 - loss: 0.2968 - val_accuracy: 0.8804 - val_loss: 0.2935
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9137 - loss: 0.2525 - val_accuracy: 0.8906 - val_loss: 0.2741
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9211 - loss: 0.2312 - val_accuracy: 0.8909 - val_loss: 0.2785
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9346 - loss: 0.1981 - val_accuracy: 0.8889 - val_loss: 0.2898
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9407 - loss: 0.1742 - val_accuracy: 0.8867 - val_loss: 0.3068
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9464 - loss: 0.1592 - val_accuracy: 0.8868 - val_loss: 0.3335
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9547 - loss: 0.1333 - val_accuracy: 0.8869 - val_loss: 0.3431
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9579 - loss: 0.1240 - val_accuracy: 0.8842 - val_loss: 0.3620
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9605 - loss: 0.1171 - val_accuracy: 0.8860 - val_loss: 0.3708
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9631 - loss: 0.1056 - val_accuracy: 0.8848 - val_loss: 0.3851
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9673 - loss: 0.0980 - val_accuracy: 0.8843 - val_loss: 0.4266
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9699 - loss: 0.0878 - val_accuracy: 0.8831 - val_loss: 0.4414
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9726 - loss: 0.0831 - val_accuracy: 0.8814 - val_loss: 0.4446
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9782 - loss: 0.0748 - val_accuracy: 0.8839 - val_loss: 0.5105
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9766 - loss: 0.0769 - val_accuracy: 0.8813 - val_loss: 0.5062
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9789 - loss: 0.0691 - val_accuracy: 0.8817 - val_loss: 0.5357
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_1, train_2, epochs=8, batch_size=512)
final_result_Dropout = model_Dropout.evaluate(test_1, test_2)
final_result_Dropout
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9282 - loss: 0.2641
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9408 - loss: 0.2077
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9527 - loss: 0.1677
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9534 - loss: 0.1584
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9606 - loss: 0.1373
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9600 - loss: 0.1365
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9603 - loss: 0.1342
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9621 - loss: 0.1191
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8718 - loss: 0.4970
[0.4913085103034973, 0.8727200031280518]
```

The loss on the test set is 0.4913 and accuracy is 87.27%.

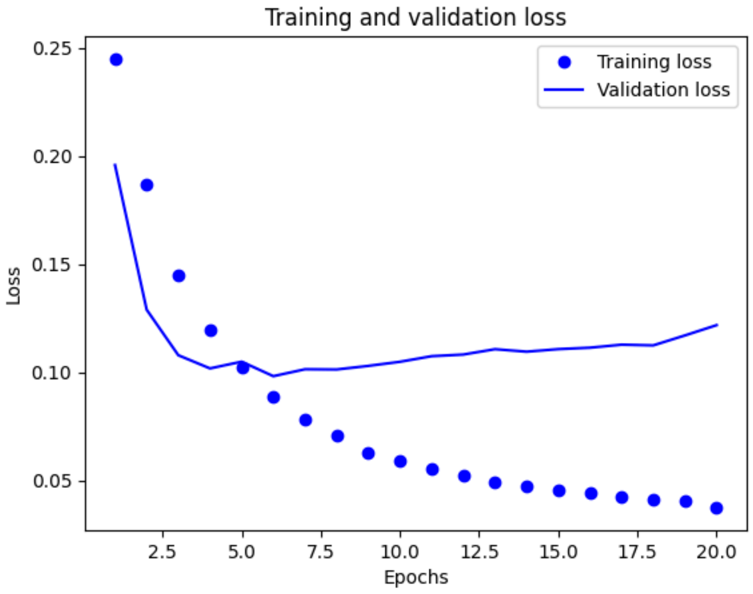
Training model with hyper tuned parameters

```
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_train_1,
                    partial_train_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_Hyper = history_model_Hyper.history
history_dict_Hyper.keys()
```

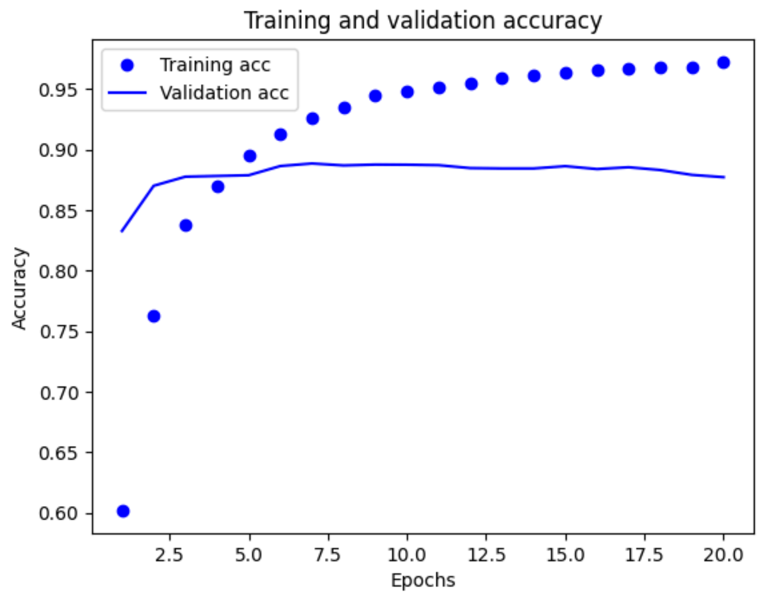
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 133ms/step - accuracy: 0.5561 - loss: 0.2552 - val_accuracy: 0.8327 - val_loss: 0.195
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.7422 - loss: 0.1977 - val_accuracy: 0.8700 - val_loss: 0.1289
Epoch 3/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.8255 - loss: 0.1518 - val_accuracy: 0.8775 - val_loss: 0.1079
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8592 - loss: 0.1270 - val_accuracy: 0.8781 - val_loss: 0.1018
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.8945 - loss: 0.1038 - val_accuracy: 0.8787 - val_loss: 0.1049
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9066 - loss: 0.0918 - val_accuracy: 0.8863 - val_loss: 0.0982
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9295 - loss: 0.0764 - val_accuracy: 0.8884 - val_loss: 0.1014
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9348 - loss: 0.0704 - val_accuracy: 0.8868 - val_loss: 0.1013
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9455 - loss: 0.0627 - val_accuracy: 0.8875 - val_loss: 0.1030
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9501 - loss: 0.0581 - val_accuracy: 0.8874 - val_loss: 0.1049
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9518 - loss: 0.0544 - val_accuracy: 0.8870 - val_loss: 0.1075
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9536 - loss: 0.0528 - val_accuracy: 0.8846 - val_loss: 0.1082
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9614 - loss: 0.0474 - val_accuracy: 0.8843 - val_loss: 0.1107
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9612 - loss: 0.0478 - val_accuracy: 0.8843 - val_loss: 0.1095
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9651 - loss: 0.0443 - val_accuracy: 0.8862 - val_loss: 0.1107
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9647 - loss: 0.0444 - val_accuracy: 0.8838 - val_loss: 0.1114
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9677 - loss: 0.0430 - val_accuracy: 0.8853 - val_loss: 0.1128
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9672 - loss: 0.0408 - val_accuracy: 0.8830 - val_loss: 0.1124
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9683 - loss: 0.0401 - val_accuracy: 0.8790 - val_loss: 0.1171
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9733 - loss: 0.0365 - val_accuracy: 0.8771 - val_loss: 0.1218
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
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()
```



```
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()
```



```
model_Hyper.fit(train_1, train_2, epochs=8, batch_size=512)
final_result_Hyper = model_Hyper.evaluate(test_1, test_2)
final_result_Hyper
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.9276 - loss: 0.0744
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9381 - loss: 0.0658
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9417 - loss: 0.0616
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9466 - loss: 0.0578
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9512 - loss: 0.0545
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9548 - loss: 0.0514
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9589 - loss: 0.0488
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9570 - loss: 0.0489
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8791 - loss: 0.1157
[0.11413271725177765, 0.8808799982070923]
```

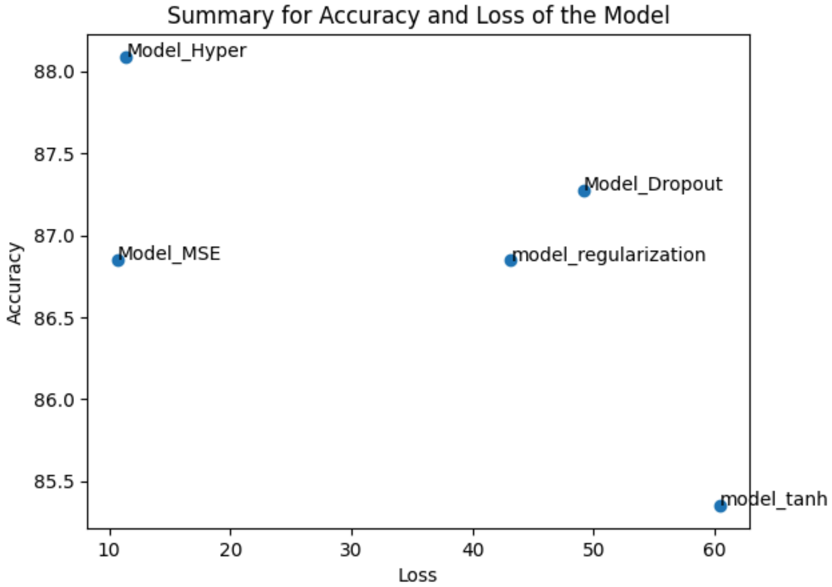
Summary

```
All_Models_Loss= np.array([final_result_Dropout[0],final_result_Hyper[0],final_result_MSE[0],final_result_regulari
All_Models_Loss
All_Models_Accuracy= np.array([final_result_Dropout[1],final_result_Hyper[1],final_result_MSE[1],final_result_regu
All_Models_Accuracy
Labels=['Model_Dropout','Model_Hyper','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(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()
```



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.

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.