This study aims to investigate multiple methods for enhancing a neural network model which operates on the IMDb dataset. We modify a pre-existing neural network framework through multiple network configuration elements. Optimization involves various elements such as hidden layer counts alongside unit numbers in each layer combined with selected loss functions and activation functions along with dropout regularization implementation.

We used the IMDb database which has positive and negative movie reviews. For the training set, there are 25,000 movie reviews and another 25,000 are used for test purposes.

```
from numpy.random import seed
seed(123)
from tensorflow.keras.datasets import imdb
(train_set, labels_train), (test_set, labels_test) = imdb.load_data(
    num_words=10000)
```

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

```
train_set
```

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

```
labels_train[0]
```

```
1
```

```
len(labels_train)
```

```
25000
```

```
test_set
```

```
array([list([1, 591, 202, 14, 31, 6, 717, 10, 10, 2, 2, 5, 4, 360, 7, 4, 177, 5760, 394, 354, 4, 123, 9, 1035, 1035,
1035, 10, 10, 13, 92, 124, 89, 488, 7944, 100, 28, 1668, 14, 31, 23, 27, 7479, 29, 220, 468, 8, 124, 14, 286, 170, 8,
157, 46, 5, 27, 239, 16, 179, 2, 38, 32, 25, 7944, 451, 202, 14, 6, 717]),
       list([1, 14, 22, 3443, 6, 176, 7, 5063, 88, 12, 2679, 23, 1310, 5, 109, 943, 4, 114, 9, 55, 606, 5, 111, 7, 4,
139, 193, 273, 23, 4, 172, 270, 11, 7216, 2, 4, 8463, 2801, 109, 1603, 21, 4, 22, 3861, 8, 6, 1193, 1330, 10, 10, 4,
105, 987, 35, 841, 2, 19, 861, 1074, 5, 1987, 2, 45, 55, 221, 15, 670, 5304, 526, 14, 1069, 4, 405, 5, 2438, 7, 27, 85,
```

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

```
labels_test[0]
```

```
0
```

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

```
9999
```

** Reviews to text**

```
word_index = imdb.get_word_index()
reversed_word_map = dict(
    [(value, key) for (key, value) in word_index.items()])
review_content = " ".join(
    [reversed_word_map.get(i - 3, "?") for i in train_set[0]])
```

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

```
review_content
```

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

Data preparation

```
import numpy as np
def vectorize_input_sequences(input_sequences, vocab_size=10000):
```

```
    binary_matrix = np.zeros((len(input_sequences), vocab_size))
    for i, sequence in enumerate(input_sequences):
        for j in sequence:
            binary_matrix[i, j] = 1.
    return binary_matrix
```

Data Vectorization

```
train_data_1 = vectorize_input_sequences(train_set)
test_data_1 = vectorize_input_sequences(test_set)
```

```
train_data_1[0]
```

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

```
test_data_1[0]
```

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

Label Vectorization

```
train_data_2 = np.asarray(labels_train).astype("float32")
test_data_2 = np.asarray(labels_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"])
```

```
seed(123)
x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]
y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]
```

```
seed(123)
history = model.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 73ms/step - accuracy: 0.6917 - loss: 0.6014 - val_accuracy: 0.8648 - val_loss: 0.3986
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.8890 - loss: 0.3504 - val_accuracy: 0.8856 - val_loss: 0.3154
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9245 - loss: 0.2510 - val_accuracy: 0.8881 - val_loss: 0.2872
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9369 - loss: 0.2021 - val_accuracy: 0.8799 - val_loss: 0.2975
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.9453 - loss: 0.1705 - val_accuracy: 0.8877 - val_loss: 0.2775
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.9577 - loss: 0.1431 - val_accuracy: 0.8784 - val_loss: 0.3046
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.9666 - loss: 0.1229 - val_accuracy: 0.8842 - val_loss: 0.2922
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 73ms/step - accuracy: 0.9716 - loss: 0.1005 - val_accuracy: 0.8831 - val_loss: 0.3122
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - accuracy: 0.9764 - loss: 0.0926 - val_accuracy: 0.8800 - val_loss: 0.3226
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9745 - loss: 0.0842 - val_accuracy: 0.8735 - val_loss: 0.3756
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - accuracy: 0.9794 - loss: 0.0722 - val_accuracy: 0.8668 - val_loss: 0.3814
```

```
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9870 - loss: 0.0608 - val_accuracy: 0.8775 - val_loss: 0.3693
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 74ms/step - accuracy: 0.9881 - loss: 0.0527 - val_accuracy: 0.8715 - val_loss: 0.3980
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 78ms/step - accuracy: 0.9905 - loss: 0.0458 - val_accuracy: 0.8753 - val_loss: 0.4098
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9937 - loss: 0.0372 - val_accuracy: 0.8608 - val_loss: 0.4778
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - accuracy: 0.9947 - loss: 0.0315 - val_accuracy: 0.8728 - val_loss: 0.4526
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9965 - loss: 0.0258 - val_accuracy: 0.8719 - val_loss: 0.4712
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - accuracy: 0.9971 - loss: 0.0228 - val_accuracy: 0.8686 - val_loss: 0.4980
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9977 - loss: 0.0204 - val_accuracy: 0.8700 - val_loss: 0.5126
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 81ms/step - accuracy: 0.9985 - loss: 0.0169 - val_accuracy: 0.8684 - val_loss: 0.5342
```

The initial training phase yielded a loss of 0.6014 with an accuracy of 69.17% on the training data, while the validation set reported 0.3986 loss and an accuracy of 86.48% at Epoch 1.

As the training progressed, the model's accuracy steadily increased, reaching 99.85% accuracy with a loss of 0.0169 by Epoch 20. However, validation performance did not improve consistently, with the model achieving a final validation accuracy of 86.84% but experiencing a higher validation loss of 0.5342.

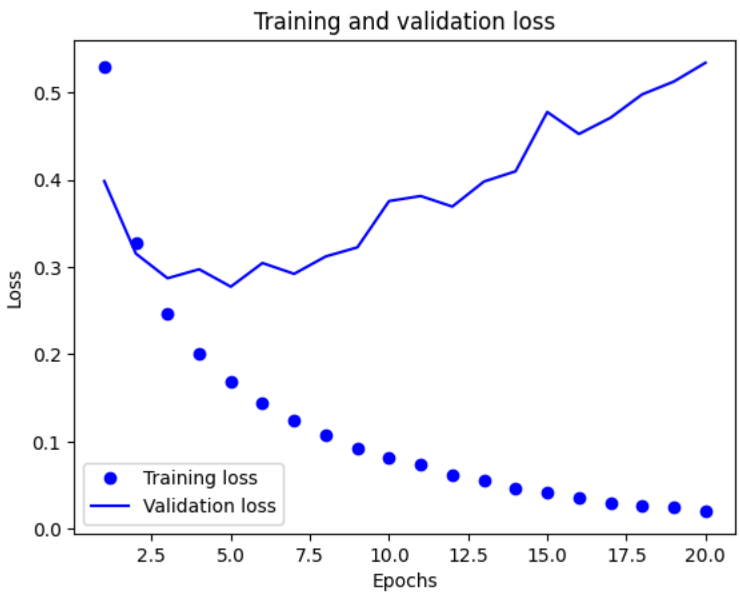
These results indicate a clear case of overfitting, where the model has learned the training data exceptionally well but struggles to generalize to new data. The increasing validation loss suggests that the model may benefit from additional regularization techniques, such as higher dropout rates or early stopping, to enhance generalization.

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

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

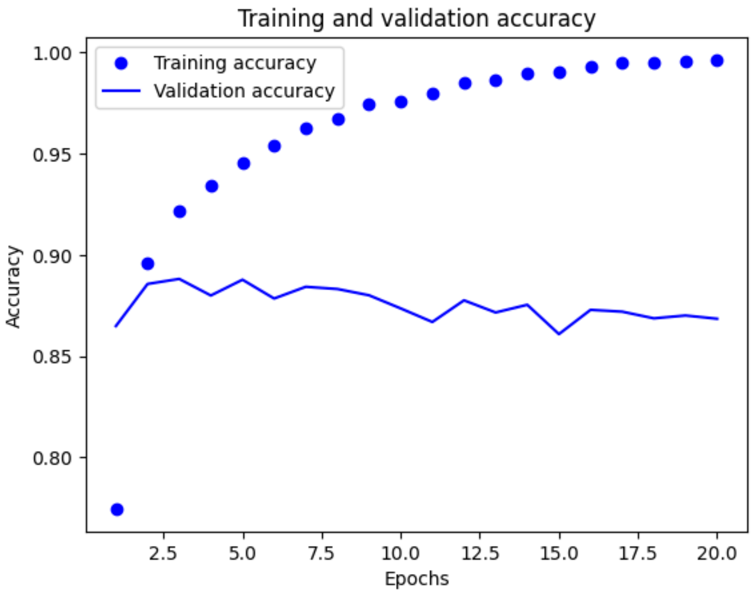
Plotting the training and validation loss

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




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

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



The two graphs suggest that overfitting the training data makes the model less good at predicting new data after a certain epoch. However, to improve the binary_matrix of the model, it may be necessary to carry out more work on the object of analysis like changing the hyperparameters of the model or using techniques like regularization.

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_data_1, train_data_2, epochs=4, batch_size=512)
binary_matrix = model.evaluate(test_data_1, test_data_2)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.7373 - loss: 0.5497
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9039 - loss: 0.2815
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9266 - loss: 0.2117
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 39ms/step - accuracy: 0.9390 - loss: 0.1786
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8832 - loss: 0.2908
```

```
binary_matrix
```

```
[0.287300169467926, 0.884880006313324]
```

During testing of the neural network model it reached 88.48% accuracy with 0.2873 loss. The model applies successfully to new data points despite continuing signs of overfitting from its training results.

```
model.predict(test_data_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.23850362],
       [0.99984056],
       [0.91147274],
       ...,
       [0.14159013],
       [0.08275776],
       [0.6641936 ]], dtype=float32)
```

Building a neural network with 1 hidden layer

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

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

x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]

y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]


history1 = model1.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

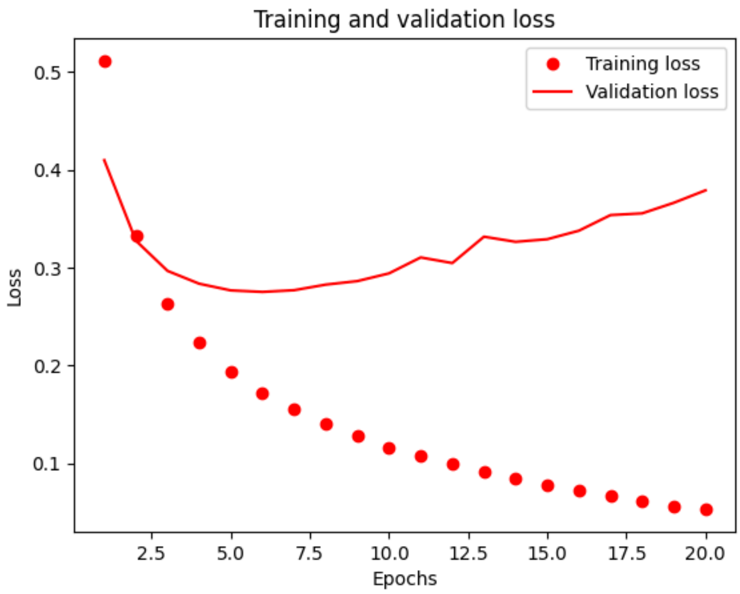
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 95ms/step - accuracy: 0.7114 - loss: 0.5833 - val_accuracy: 0.8585 - val_loss: 0.4098
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.8998 - loss: 0.3476 - val_accuracy: 0.8843 - val_loss: 0.3275
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9154 - loss: 0.2678 - val_accuracy: 0.8881 - val_loss: 0.2966
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9315 - loss: 0.2211 - val_accuracy: 0.8880 - val_loss: 0.2835
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9443 - loss: 0.1906 - val_accuracy: 0.8893 - val_loss: 0.2767
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9503 - loss: 0.1699 - val_accuracy: 0.8871 - val_loss: 0.2751
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9551 - loss: 0.1566 - val_accuracy: 0.8858 - val_loss: 0.2769
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9599 - loss: 0.1385 - val_accuracy: 0.8860 - val_loss: 0.2826
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.9641 - loss: 0.1270 - val_accuracy: 0.8850 - val_loss: 0.2862
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.9693 - loss: 0.1154 - val_accuracy: 0.8815 - val_loss: 0.2941
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9714 - loss: 0.1082 - val_accuracy: 0.8809 - val_loss: 0.3104
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9746 - loss: 0.0966 - val_accuracy: 0.8810 - val_loss: 0.3046
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9800 - loss: 0.0864 - val_accuracy: 0.8766 - val_loss: 0.3316
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9806 - loss: 0.0799 - val_accuracy: 0.8761 - val_loss: 0.3263
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9826 - loss: 0.0774 - val_accuracy: 0.8794 - val_loss: 0.3290
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9864 - loss: 0.0697 - val_accuracy: 0.8771 - val_loss: 0.3377
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9883 - loss: 0.0634 - val_accuracy: 0.8773 - val_loss: 0.3537
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9884 - loss: 0.0623 - val_accuracy: 0.8760 - val_loss: 0.3554
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 54ms/step - accuracy: 0.9916 - loss: 0.0550 - val_accuracy: 0.8754 - val_loss: 0.3662
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9921 - loss: 0.0528 - val_accuracy: 0.8743 - val_loss: 0.3789
```

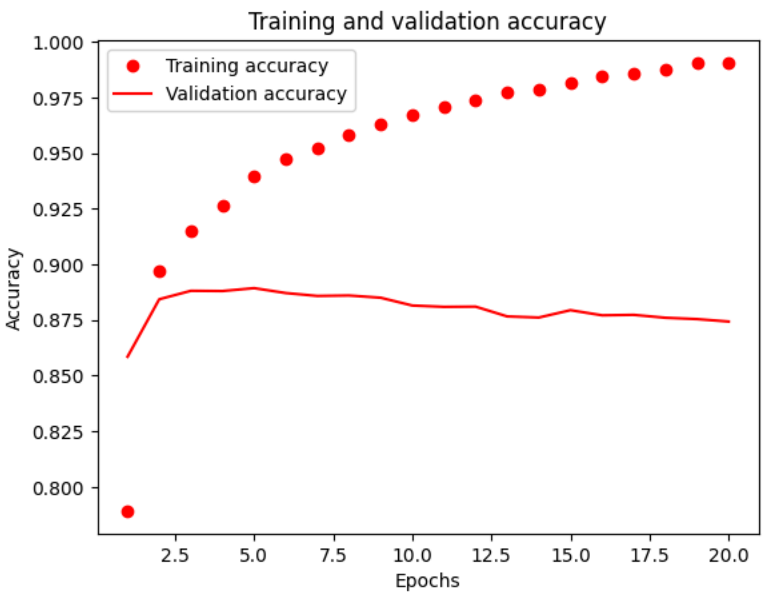
```
history_dict = history1.history
history_dict.keys()
```

```
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_data_1, train_data_2, epochs=5, batch_size=512)
binary_matrix1 = model1.evaluate(test_data_1, test_data_2)
```

```
Epoch 1/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.7414 - loss: 0.5413
Epoch 2/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.8995 - loss: 0.2990
Epoch 3/5
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9232 - loss: 0.2309
Epoch 4/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 36ms/step - accuracy: 0.9320 - loss: 0.1994
Epoch 5/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9410 - loss: 0.1759
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8863 - loss: 0.2816
```

```
binary_matrix1
```

```
[0.2796546220779419, 0.8884400129318237]
```

The test set achieved a loss of 0.2830 and an accuracy of 88.62%.

```
model1.predict(test_data_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.21907678],
       [0.99987614],
       [0.8446142 ],
       ...,
       [0.1310835 ],
       [0.09292865],
       [0.5650619 ]], dtype=float32)
```

Creating a neural network with three hidden layers

```
np.random.seed(123)
model_3 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_3.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]

y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]

history3 = model_3.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

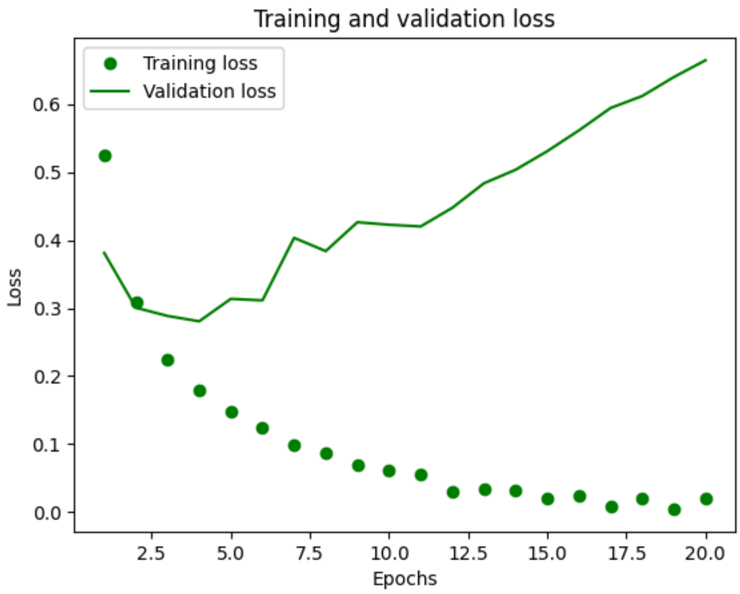
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 67ms/step - accuracy: 0.6993 - loss: 0.5994 - val_accuracy: 0.8694 - val_loss: 0.3811
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.8929 - loss: 0.3240 - val_accuracy: 0.8876 - val_loss: 0.3006
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9289 - loss: 0.2218 - val_accuracy: 0.8845 - val_loss: 0.2885
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9459 - loss: 0.1707 - val_accuracy: 0.8852 - val_loss: 0.2806
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9475 - loss: 0.1504 - val_accuracy: 0.8819 - val_loss: 0.3136
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9636 - loss: 0.1197 - val_accuracy: 0.8849 - val_loss: 0.3114
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9707 - loss: 0.0955 - val_accuracy: 0.8581 - val_loss: 0.4033
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9703 - loss: 0.0909 - val_accuracy: 0.8692 - val_loss: 0.3838
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9793 - loss: 0.0695 - val_accuracy: 0.8670 - val_loss: 0.4263
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9801 - loss: 0.0635 - val_accuracy: 0.8712 - val_loss: 0.4226
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9893 - loss: 0.0441 - val_accuracy: 0.8759 - val_loss: 0.4201
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9952 - loss: 0.0279 - val_accuracy: 0.8739 - val_loss: 0.4475
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9925 - loss: 0.0297 - val_accuracy: 0.8704 - val_loss: 0.4836
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9952 - loss: 0.0246 - val_accuracy: 0.8727 - val_loss: 0.5035
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9990 - loss: 0.0136 - val_accuracy: 0.8726 - val_loss: 0.5308
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9990 - loss: 0.0113 - val_accuracy: 0.8702 - val_loss: 0.5614
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - accuracy: 0.9998 - loss: 0.0075 - val_accuracy: 0.8692 - val_loss: 0.5943
Epoch 18/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9967 - loss: 0.0147 - val_accuracy: 0.8702 - val_loss: 0.6119
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9999 - loss: 0.0046 - val_accuracy: 0.8700 - val_loss: 0.6398
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9956 - loss: 0.0163 - val_accuracy: 0.8700 - val_loss: 0.6645
```

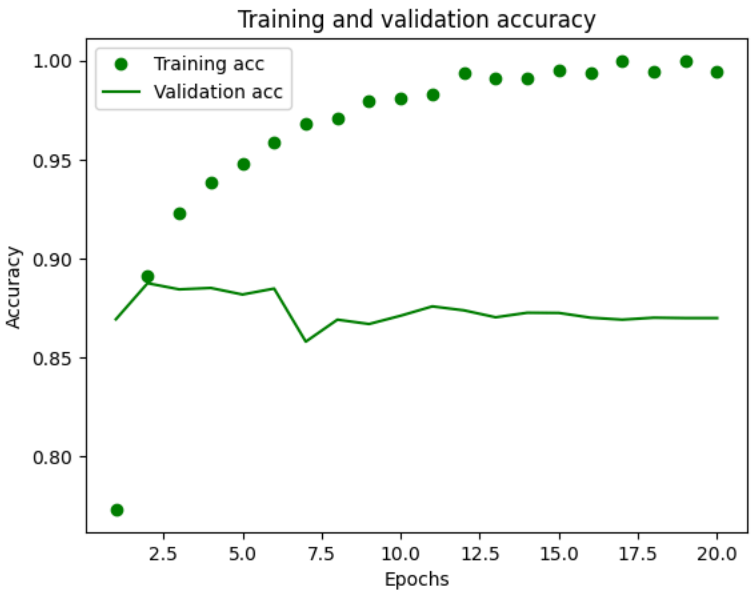
```
history_dict3 = history3.history
history_dict3.keys()
```

```
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_data_1, train_data_2, epochs=3, batch_size=512)
binary_matrix_3 = model_3.evaluate(test_data_1, test_data_2)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.7286 - loss: 0.5619
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 40ms/step - accuracy: 0.9010 - loss: 0.2735
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9309 - loss: 0.1988
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8709 - loss: 0.3213
```

The test set has a loss of 0.28 and an accuracy of 88.59%.

```
binary_matrix_3
```

```
[0.31690192222595215, 0.8728799819946289]
```

```
model_3.predict(test_data_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step
array([[0.374492  ],
       [0.9997954 ],
       [0.9836259 ],
       ...,
       [0.19048955],
       [0.190435  ],
       [0.74117744]], dtype=float32)
```

Building Neural Network with 32 units.

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

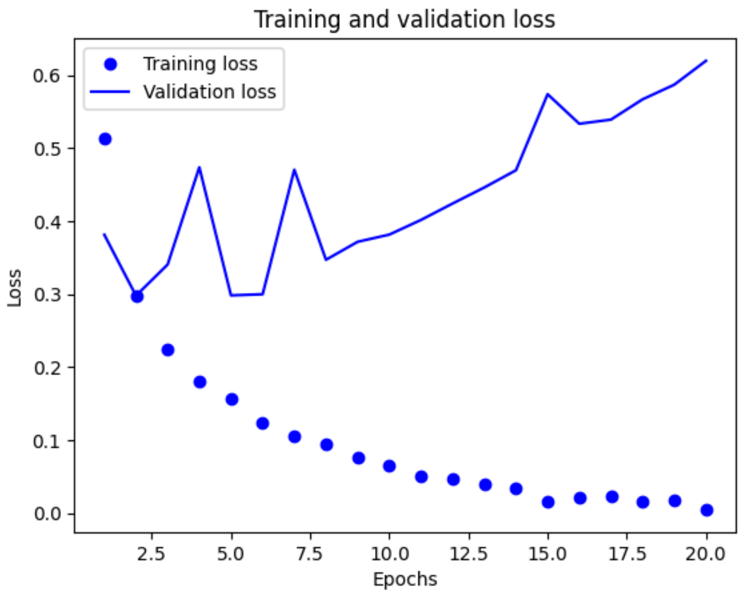
```
#model validation
x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]

y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]

np.random.seed(123)
history32 = model_32.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 77ms/step - accuracy: 0.6901 - loss: 0.5961 - val_accuracy: 0.8534 - val_loss: 0.3812
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.8933 - loss: 0.3135 - val_accuracy: 0.8850 - val_loss: 0.2978
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9175 - loss: 0.2302 - val_accuracy: 0.8598 - val_loss: 0.3405
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9377 - loss: 0.1814 - val_accuracy: 0.8177 - val_loss: 0.4736
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 58ms/step - accuracy: 0.9412 - loss: 0.1624 - val_accuracy: 0.8840 - val_loss: 0.2982
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9639 - loss: 0.1197 - val_accuracy: 0.8858 - val_loss: 0.2998
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9698 - loss: 0.1019 - val_accuracy: 0.8372 - val_loss: 0.4703
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 45ms/step - accuracy: 0.9634 - loss: 0.1008 - val_accuracy: 0.8740 - val_loss: 0.3470
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9803 - loss: 0.0707 - val_accuracy: 0.8718 - val_loss: 0.3716
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9815 - loss: 0.0621 - val_accuracy: 0.8731 - val_loss: 0.3814
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9876 - loss: 0.0502 - val_accuracy: 0.8731 - val_loss: 0.4015
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 54ms/step - accuracy: 0.9873 - loss: 0.0442 - val_accuracy: 0.8780 - val_loss: 0.4240
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 0.9933 - loss: 0.0320 - val_accuracy: 0.8747 - val_loss: 0.4459
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.9966 - loss: 0.0224 - val_accuracy: 0.8742 - val_loss: 0.4695
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 45ms/step - accuracy: 0.9985 - loss: 0.0168 - val_accuracy: 0.8664 - val_loss: 0.5736
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9903 - loss: 0.0349 - val_accuracy: 0.8733 - val_loss: 0.5332
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - accuracy: 0.9927 - loss: 0.0278 - val_accuracy: 0.8722 - val_loss: 0.5388
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - accuracy: 0.9981 - loss: 0.0114 - val_accuracy: 0.8747 - val_loss: 0.5667
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 45ms/step - accuracy: 0.9992 - loss: 0.0076 - val_accuracy: 0.8734 - val_loss: 0.5867
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9999 - loss: 0.0055 - val_accuracy: 0.8735 - val_loss: 0.6193
```

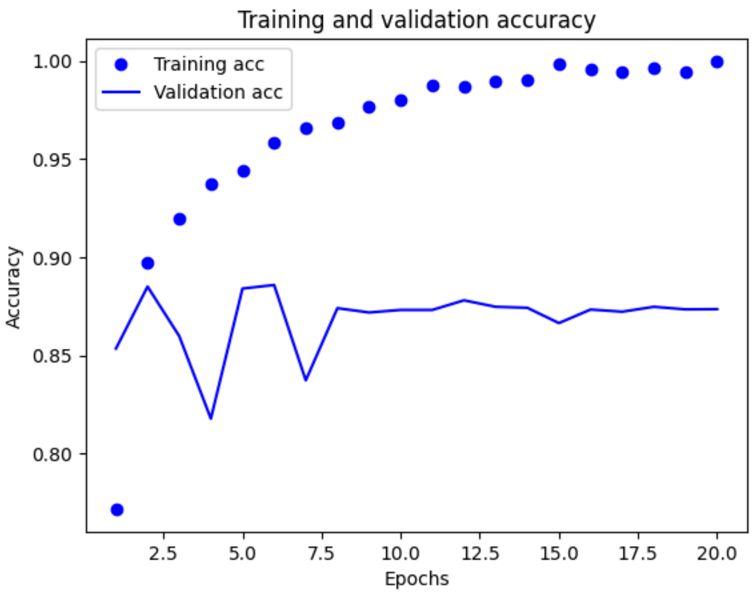
```
history_dict32 = history32.history
history_dict32.keys()
```

```
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()
```




```
history_32 = model_32.fit(train_data_1, train_data_2, epochs=3, batch_size=12)
final_result_32 = model_32.evaluate(test_data_1, test_data_2)
print(final_result_32)
```

```
Epoch 1/3
2084/2084 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.9335 - loss: 0.2234
Epoch 2/3
2084/2084 ━━━━━━━━━━━━━━━━━━━━ 13s 6ms/step - accuracy: 0.9421 - loss: 0.1674
Epoch 3/3
2084/2084 ━━━━━━━━━━━━━━━━━━━━ 20s 6ms/step - accuracy: 0.9532 - loss: 0.1354
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8734 - loss: 0.3742
[0.3669483959674835, 0.8775200247764587]
```

```
model_32.predict(test_data_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
array([[0.0996104 ],
       [1.        ],
       [0.9991016 ],
       ...,
```

```
       [0.09804397],
       [0.04331056],
       [0.641288  ]], dtype=float32)
```

Training the model with 64 units

```
np.random.seed(123)
model_64 = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_64.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]

y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]

np.random.seed(123)
history64 = model_64.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

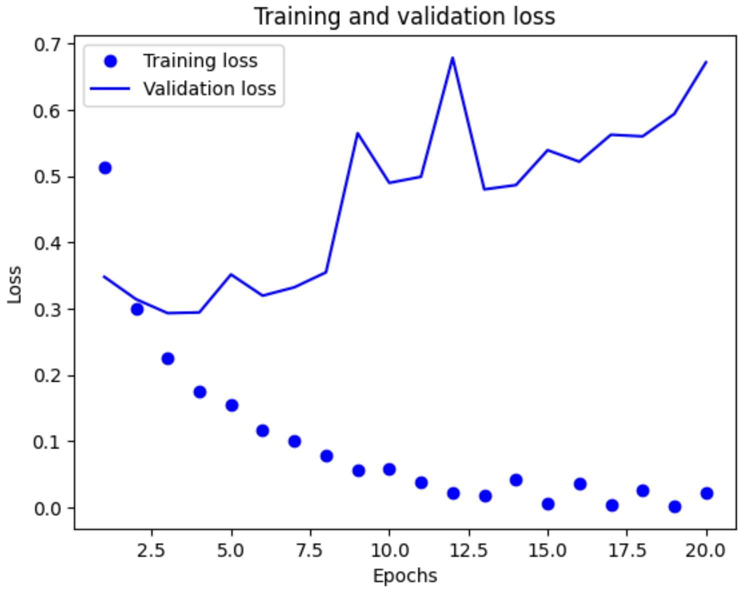
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 102ms/step - accuracy: 0.6751 - loss: 0.5910 - val_accuracy: 0.8710 - val_loss: 0.3480
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 70ms/step - accuracy: 0.8933 - loss: 0.3001 - val_accuracy: 0.8714 - val_loss: 0.3143
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9174 - loss: 0.2249 - val_accuracy: 0.8795 - val_loss: 0.2933
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 109ms/step - accuracy: 0.9392 - loss: 0.1774 - val_accuracy: 0.8842 - val_loss: 0.2944
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9426 - loss: 0.1522 - val_accuracy: 0.8626 - val_loss: 0.3516
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9628 - loss: 0.1114 - val_accuracy: 0.8772 - val_loss: 0.3196
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9715 - loss: 0.0922 - val_accuracy: 0.8836 - val_loss: 0.3322
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 0.9784 - loss: 0.0705 - val_accuracy: 0.8822 - val_loss: 0.3548
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 102ms/step - accuracy: 0.9835 - loss: 0.0582 - val_accuracy: 0.8359 - val_loss: 0.5646
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 69ms/step - accuracy: 0.9693 - loss: 0.0797 - val_accuracy: 0.8624 - val_loss: 0.4897
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - accuracy: 0.9939 - loss: 0.0297 - val_accuracy: 0.8628 - val_loss: 0.4989
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9952 - loss: 0.0242 - val_accuracy: 0.8408 - val_loss: 0.6784
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 97ms/step - accuracy: 0.9907 - loss: 0.0333 - val_accuracy: 0.8774 - val_loss: 0.4799
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 69ms/step - accuracy: 0.9852 - loss: 0.0466 - val_accuracy: 0.8773 - val_loss: 0.4864
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - accuracy: 0.9999 - loss: 0.0064 - val_accuracy: 0.8760 - val_loss: 0.5391
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.9893 - loss: 0.0349 - val_accuracy: 0.8768 - val_loss: 0.5217
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 103ms/step - accuracy: 0.9999 - loss: 0.0039 - val_accuracy: 0.8764 - val_loss: 0.5623
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 71ms/step - accuracy: 0.9955 - loss: 0.0157 - val_accuracy: 0.8765 - val_loss: 0.5599
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.9999 - loss: 0.0026 - val_accuracy: 0.8765 - val_loss: 0.5937
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.9993 - loss: 0.0038 - val_accuracy: 0.8705 - val_loss: 0.6715
```

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

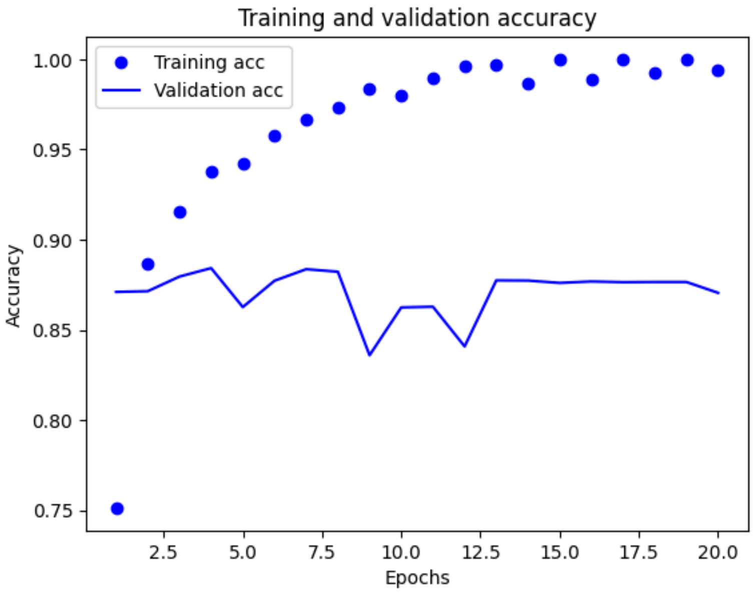
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
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_data_1, train_data_2, epochs=3, batch_size=512)
binary_matrix_64 = model_64.evaluate(test_data_1, test_data_2)
binary_matrix_64
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 67ms/step - accuracy: 0.9481 - loss: 0.1970
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9695 - loss: 0.0985
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 48ms/step - accuracy: 0.9838 - loss: 0.0554
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8593 - loss: 0.4576
[0.45947566628456116, 0.8582800030708313]
```

```
model_64.predict(test_data_1)
```

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

```
array([[0.02172247],
       [0.99999917],
       [0.08185229],
       ...,
       [0.0206599 ],
       [0.00261007],
       [0.2611061 ]], dtype=float32)
```

The validation set has an accuracy of 85.82%.

Training the model with 128 units

```
np.random.seed(123)
model_128 = keras.Sequential([
    layers.Dense(128, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_128.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]

y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]

np.random.seed(123)
history128 = model_128.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

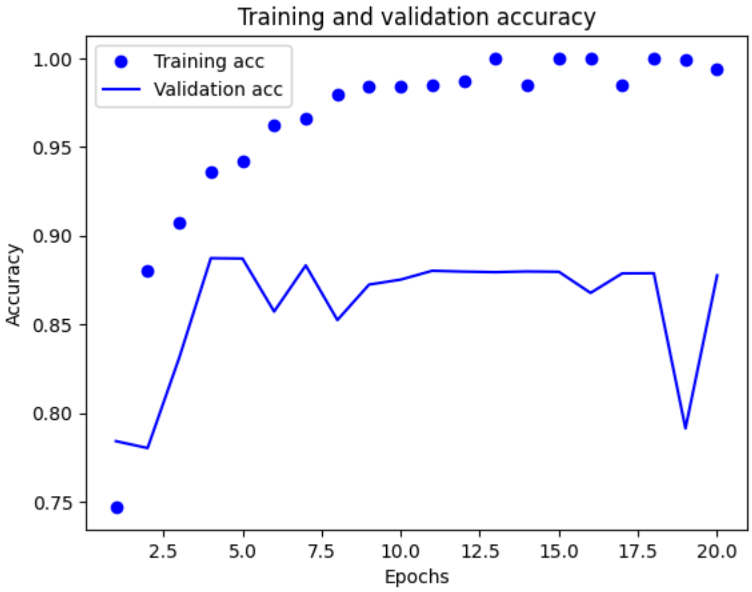
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 199ms/step - accuracy: 0.6666 - loss: 0.6052 - val_accuracy: 0.7841 - val_loss: 0.4591
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 100ms/step - accuracy: 0.8734 - loss: 0.3213 - val_accuracy: 0.7802 - val_loss: 0.5050
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 130ms/step - accuracy: 0.8985 - loss: 0.2536 - val_accuracy: 0.8311 - val_loss: 0.4165
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 111ms/step - accuracy: 0.9358 - loss: 0.1729 - val_accuracy: 0.8873 - val_loss: 0.2773
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 119ms/step - accuracy: 0.9482 - loss: 0.1469 - val_accuracy: 0.8871 - val_loss: 0.2831
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - accuracy: 0.9689 - loss: 0.1027 - val_accuracy: 0.8572 - val_loss: 0.3897
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 132ms/step - accuracy: 0.9663 - loss: 0.0984 - val_accuracy: 0.8832 - val_loss: 0.3270
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 100ms/step - accuracy: 0.9806 - loss: 0.0607 - val_accuracy: 0.8524 - val_loss: 0.4577
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 135ms/step - accuracy: 0.9660 - loss: 0.0902 - val_accuracy: 0.8724 - val_loss: 0.4117
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 98ms/step - accuracy: 0.9725 - loss: 0.0823 - val_accuracy: 0.8752 - val_loss: 0.4262
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 97ms/step - accuracy: 0.9859 - loss: 0.0524 - val_accuracy: 0.8802 - val_loss: 0.4169
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 127ms/step - accuracy: 0.9965 - loss: 0.0174 - val_accuracy: 0.8797 - val_loss: 0.4078
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 124ms/step - accuracy: 0.9999 - loss: 0.0078 - val_accuracy: 0.8794 - val_loss: 0.4857
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 8s 213ms/step - accuracy: 0.9945 - loss: 0.0217 - val_accuracy: 0.8798 - val_loss: 0.4557
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 106ms/step - accuracy: 0.9999 - loss: 0.0043 - val_accuracy: 0.8796 - val_loss: 0.5115
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 161ms/step - accuracy: 1.0000 - loss: 0.0024 - val_accuracy: 0.8677 - val_loss: 0.5928
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 102ms/step - accuracy: 0.9714 - loss: 0.0985 - val_accuracy: 0.8787 - val_loss: 0.5133
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 106ms/step - accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 0.8788 - val_loss: 0.5685
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 145ms/step - accuracy: 0.9999 - loss: 0.0015 - val_accuracy: 0.7913 - val_loss: 1.1907
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 120ms/step - accuracy: 0.9792 - loss: 0.0556 - val_accuracy: 0.8776 - val_loss: 0.5769
```

```
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_data_1, train_data_2, epochs=2, batch_size=512)
binary_matrix_128 = model_128.evaluate(test_data_1, test_data_2)
binary_matrix_128
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 5s 102ms/step - accuracy: 0.9455 - loss: 0.1902
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 79ms/step - accuracy: 0.9736 - loss: 0.0856
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.8626 - loss: 0.4248
[0.41691234707832336, 0.8646799921989441]
```

```
model_128.predict(test_data_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step
array([[0.02205517],
       [1.        ],
       [0.98543817],
       ...,
       [0.07473296],
       [0.0180116 ],
       [0.6767774 ]], dtype=float32)
```

The validation set has an accuracy of 86.46%.

MSE Loss Function

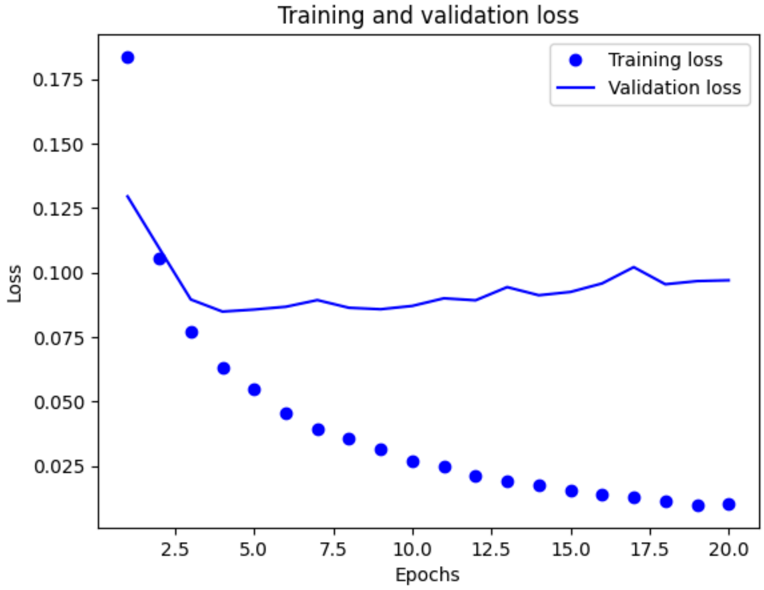
```
np.random.seed(123)
model_MSE = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
#Model compilation
model_MSE.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
# validation
x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]

y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]
# Model Fit
np.random.seed(123)
history_model_MSE = model_MSE.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 84ms/step - accuracy: 0.6707 - loss: 0.2144 - val_accuracy: 0.8631 - val_loss: 0.1296
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.8875 - loss: 0.1110 - val_accuracy: 0.8614 - val_loss: 0.1095
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9185 - loss: 0.0786 - val_accuracy: 0.8872 - val_loss: 0.0896
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9293 - loss: 0.0648 - val_accuracy: 0.8898 - val_loss: 0.0849
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9414 - loss: 0.0544 - val_accuracy: 0.8839 - val_loss: 0.0857
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9588 - loss: 0.0433 - val_accuracy: 0.8826 - val_loss: 0.0867
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9624 - loss: 0.0397 - val_accuracy: 0.8749 - val_loss: 0.0893
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9631 - loss: 0.0371 - val_accuracy: 0.8787 - val_loss: 0.0863
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9717 - loss: 0.0303 - val_accuracy: 0.8816 - val_loss: 0.0858
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9769 - loss: 0.0261 - val_accuracy: 0.8798 - val_loss: 0.0871
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9798 - loss: 0.0243 - val_accuracy: 0.8795 - val_loss: 0.0900
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9840 - loss: 0.0201 - val_accuracy: 0.8794 - val_loss: 0.0892
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9855 - loss: 0.0184 - val_accuracy: 0.8728 - val_loss: 0.0944
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9860 - loss: 0.0188 - val_accuracy: 0.8772 - val_loss: 0.0912
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9900 - loss: 0.0141 - val_accuracy: 0.8769 - val_loss: 0.0925
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9901 - loss: 0.0130 - val_accuracy: 0.8732 - val_loss: 0.0958
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9906 - loss: 0.0125 - val_accuracy: 0.8678 - val_loss: 0.1021
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9915 - loss: 0.0114 - val_accuracy: 0.8755 - val_loss: 0.0955
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9932 - loss: 0.0093 - val_accuracy: 0.8739 - val_loss: 0.0967
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.9910 - loss: 0.0102 - val_accuracy: 0.8745 - val_loss: 0.0970
```

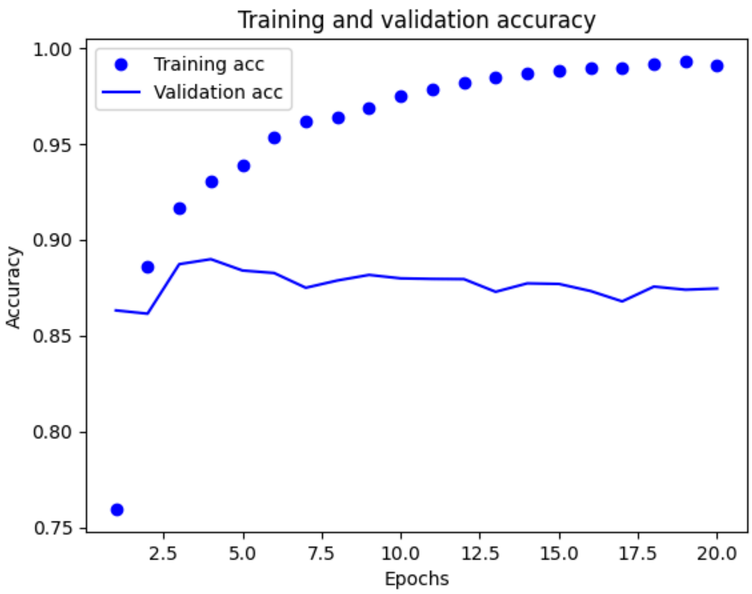
```
history_dict_MSE = history_model_MSE.history
history_dict_MSE.keys()
```

```
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_data_1, train_data_2, epochs=8, batch_size=512)
binary_matrix_MSE = model_MSE.evaluate(test_data_1, test_data_2)
binary_matrix_MSE
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9422 - loss: 0.0469
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9617 - loss: 0.0341
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9692 - loss: 0.0290
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9731 - loss: 0.0258
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.9781 - loss: 0.0224
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9773 - loss: 0.0232
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9830 - loss: 0.0182
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9836 - loss: 0.0179
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8677 - loss: 0.1084
[0.10681847482919693, 0.8680400252342224]
```

```
model_MSE.predict(test_data_1)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.01819231],
       [0.99996614],
       [0.75049436],
       ...,
       [0.04526152],
       [0.01132891],
       [0.55426615]], dtype=float32)
```

Tanh Activation Function

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

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

x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]

y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]

np.random.seed(123)

history_tanh = model_tanh.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

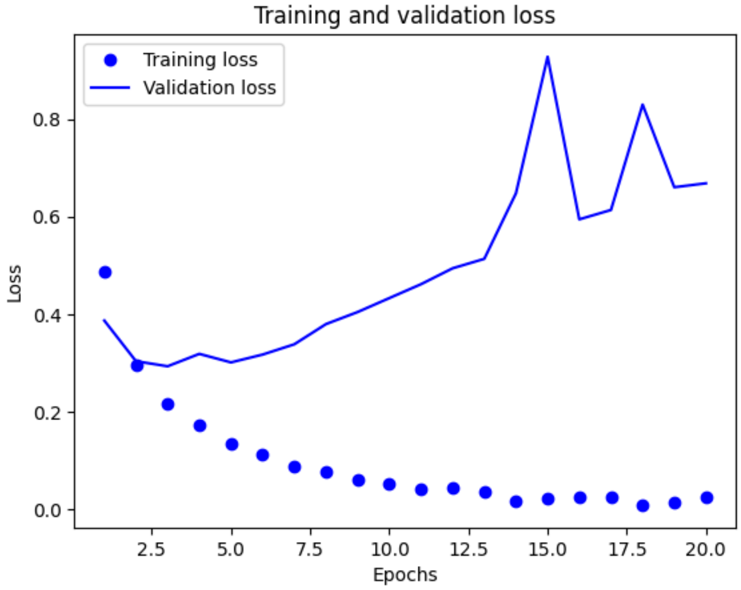
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 68ms/step - accuracy: 0.7037 - loss: 0.5736 - val_accuracy: 0.8476 - val_loss: 0.3869
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 75ms/step - accuracy: 0.9004 - loss: 0.3059 - val_accuracy: 0.8806 - val_loss: 0.3036
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.9279 - loss: 0.2226 - val_accuracy: 0.8797 - val_loss: 0.2933
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9436 - loss: 0.1688 - val_accuracy: 0.8702 - val_loss: 0.3187
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - accuracy: 0.9567 - loss: 0.1354 - val_accuracy: 0.8807 - val_loss: 0.3012
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9629 - loss: 0.1109 - val_accuracy: 0.8801 - val_loss: 0.3173
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.9772 - loss: 0.0818 - val_accuracy: 0.8805 - val_loss: 0.3383
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9781 - loss: 0.0738 - val_accuracy: 0.8758 - val_loss: 0.3797
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9845 - loss: 0.0581 - val_accuracy: 0.8718 - val_loss: 0.4044
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9895 - loss: 0.0435 - val_accuracy: 0.8716 - val_loss: 0.4330
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9925 - loss: 0.0340 - val_accuracy: 0.8727 - val_loss: 0.4614
Epoch 12/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9921 - loss: 0.0304 - val_accuracy: 0.8691 - val_loss: 0.4941
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - accuracy: 0.9963 - loss: 0.0223 - val_accuracy: 0.8669 - val_loss: 0.5134
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9984 - loss: 0.0147 - val_accuracy: 0.8462 - val_loss: 0.6478
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9879 - loss: 0.0340 - val_accuracy: 0.8169 - val_loss: 0.9275
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9792 - loss: 0.0587 - val_accuracy: 0.8646 - val_loss: 0.5944
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9965 - loss: 0.0175 - val_accuracy: 0.8658 - val_loss: 0.6136
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9996 - loss: 0.0062 - val_accuracy: 0.8351 - val_loss: 0.8295
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9883 - loss: 0.0337 - val_accuracy: 0.8635 - val_loss: 0.6602
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9968 - loss: 0.0138 - val_accuracy: 0.8648 - val_loss: 0.6682
```

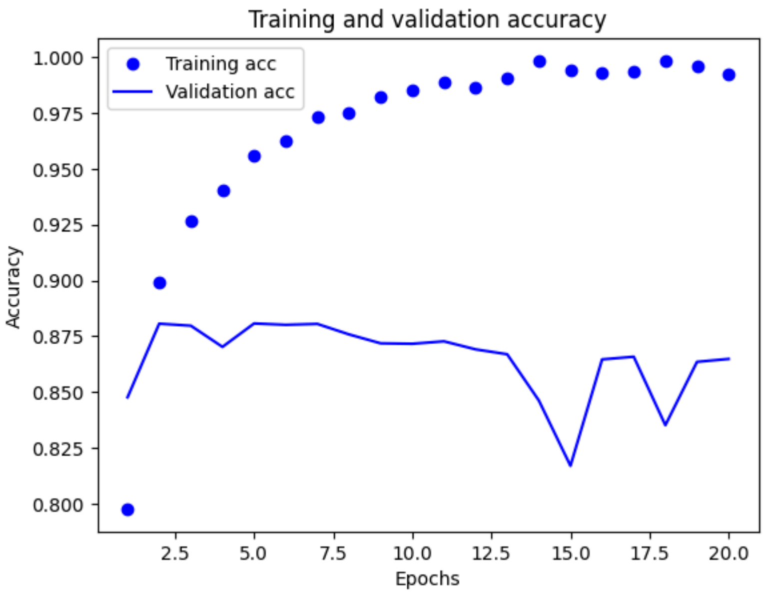
```
history_dict_tanh = history_tanh.history
history_dict_tanh.keys()
```

```
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_data_1, train_data_2, epochs=8, batch_size=512)
binary_matrix_tanh = model_tanh.evaluate(test_data_1, test_data_2)
binary_matrix_tanh
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9430 - loss: 0.2699
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9612 - loss: 0.1452
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9648 - loss: 0.1161
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9706 - loss: 0.0959
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.9789 - loss: 0.0787
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9792 - loss: 0.0687
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 40ms/step - accuracy: 0.9824 - loss: 0.0625
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9799 - loss: 0.0623
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8538 - loss: 0.5919
[0.5881032943725586, 0.8543199896812439]
```

Adam Optimizer Function

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

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

x_val = train_data_1[:10000]
partial_train_data_1 = train_data_1[10000:]

y_val = train_data_2[:10000]
partial_train_data_2 = train_data_2[10000:]

np.random.seed(123)

history_adam = model_adam.fit(partial_train_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

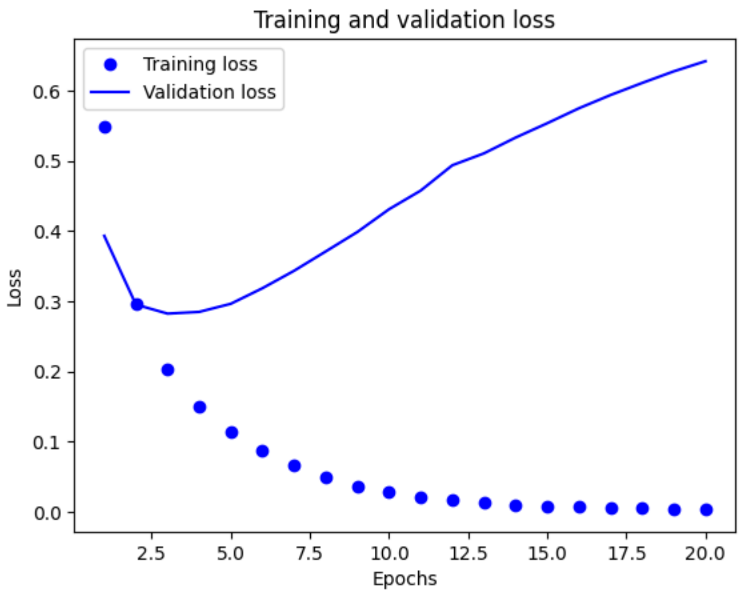
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 0.6743 - loss: 0.6239 - val_accuracy: 0.8575 - val_loss: 0.3929
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.8908 - loss: 0.3210 - val_accuracy: 0.8882 - val_loss: 0.2950
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9358 - loss: 0.2048 - val_accuracy: 0.8865 - val_loss: 0.2821
```

```
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9538 - loss: 0.1537 - val_accuracy: 0.8864 - val_loss: 0.2845
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.9678 - loss: 0.1166 - val_accuracy: 0.8843 - val_loss: 0.2961
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9809 - loss: 0.0847 - val_accuracy: 0.8811 - val_loss: 0.3183
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9860 - loss: 0.0666 - val_accuracy: 0.8802 - val_loss: 0.3431
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9915 - loss: 0.0504 - val_accuracy: 0.8771 - val_loss: 0.3707
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9950 - loss: 0.0365 - val_accuracy: 0.8761 - val_loss: 0.3984
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9969 - loss: 0.0286 - val_accuracy: 0.8730 - val_loss: 0.4308
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.9990 - loss: 0.0205 - val_accuracy: 0.8729 - val_loss: 0.4574
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9995 - loss: 0.0153 - val_accuracy: 0.8695 - val_loss: 0.4935
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9997 - loss: 0.0128 - val_accuracy: 0.8701 - val_loss: 0.5104
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9998 - loss: 0.0102 - val_accuracy: 0.8705 - val_loss: 0.5329
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9997 - loss: 0.0083 - val_accuracy: 0.8700 - val_loss: 0.5533
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9998 - loss: 0.0066 - val_accuracy: 0.8685 - val_loss: 0.5746
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 1.0000 - loss: 0.0055 - val_accuracy: 0.8684 - val_loss: 0.5935
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9999 - loss: 0.0047 - val_accuracy: 0.8680 - val_loss: 0.6107
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 1.0000 - loss: 0.0040 - val_accuracy: 0.8680 - val_loss: 0.6273
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9999 - loss: 0.0036 - val_accuracy: 0.8685 - val_loss: 0.6417
```

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

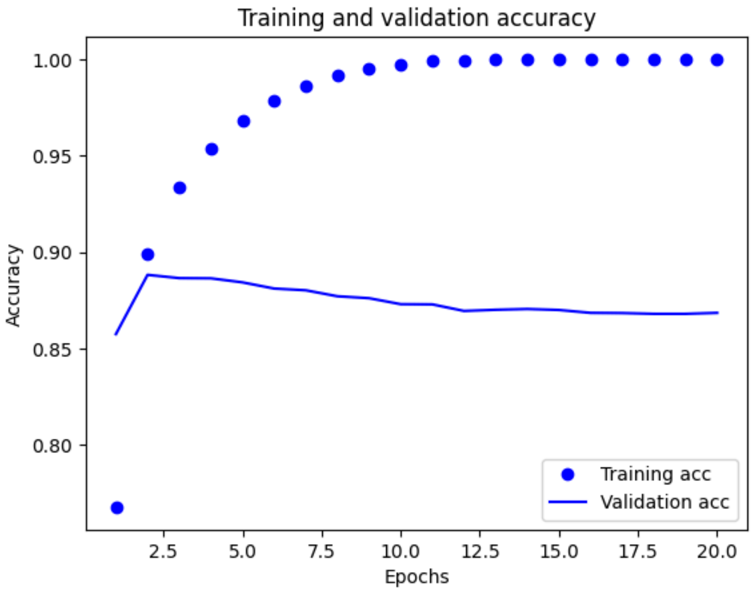
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```python
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()
```




```python
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_data_1, train_data_2, epochs=4, batch_size=512)
binary_matrix_adam = model_adam.evaluate(test_data_1, test_data_2)
binary_matrix_adam
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9436 - loss: 0.2277
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9666 - loss: 0.1105
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9809 - loss: 0.0702
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9887 - loss: 0.0507
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8592 - loss: 0.4955
[0.49056386947631836, 0.8596400022506714]
```

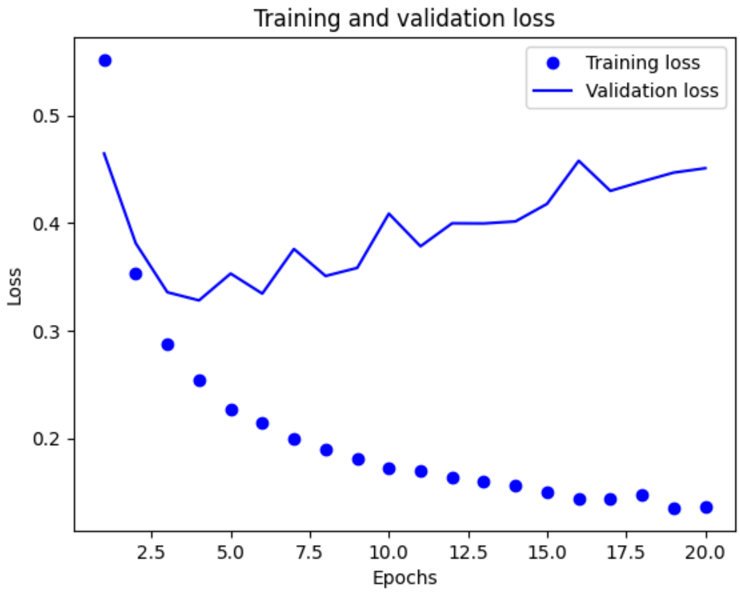
Regularization

```
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_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_regularization = history_model_regularization.history
history_dict_regularization.keys()
```

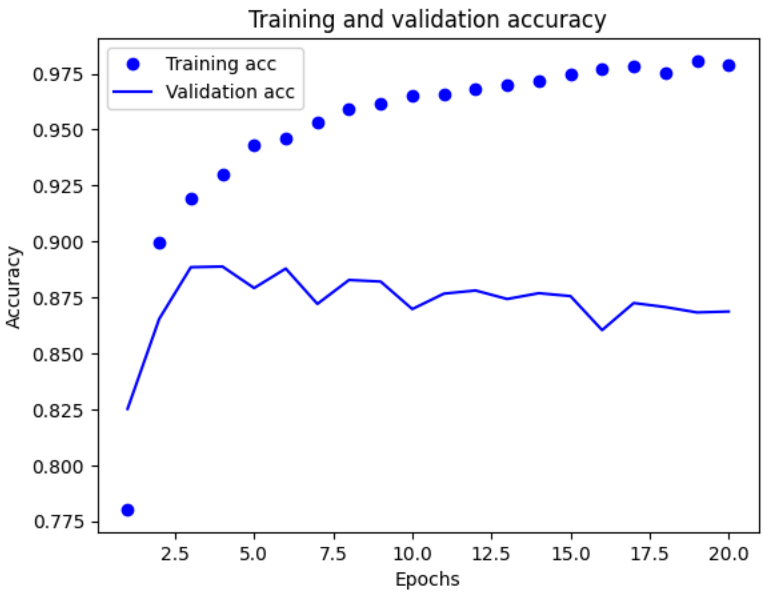
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.6936 - loss: 0.6340 - val_accuracy: 0.8252 - val_loss: 0.4646
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.8999 - loss: 0.3638 - val_accuracy: 0.8654 - val_loss: 0.3812
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 64ms/step - accuracy: 0.9167 - loss: 0.2943 - val_accuracy: 0.8885 - val_loss: 0.3356
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9327 - loss: 0.2499 - val_accuracy: 0.8888 - val_loss: 0.3281
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9478 - loss: 0.2202 - val_accuracy: 0.8792 - val_loss: 0.3530
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9504 - loss: 0.2086 - val_accuracy: 0.8879 - val_loss: 0.3344
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9588 - loss: 0.1918 - val_accuracy: 0.8721 - val_loss: 0.3759
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9631 - loss: 0.1819 - val_accuracy: 0.8828 - val_loss: 0.3508
Epoch 9/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9686 - loss: 0.1725 - val_accuracy: 0.8821 - val_loss: 0.3583
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9697 - loss: 0.1654 - val_accuracy: 0.8698 - val_loss: 0.4087
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9702 - loss: 0.1620 - val_accuracy: 0.8767 - val_loss: 0.3783
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.9754 - loss: 0.1520 - val_accuracy: 0.8781 - val_loss: 0.3997
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9749 - loss: 0.1511 - val_accuracy: 0.8743 - val_loss: 0.3996
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9799 - loss: 0.1417 - val_accuracy: 0.8769 - val_loss: 0.4015
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9800 - loss: 0.1423 - val_accuracy: 0.8756 - val_loss: 0.4178
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9806 - loss: 0.1380 - val_accuracy: 0.8604 - val_loss: 0.4579
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9813 - loss: 0.1381 - val_accuracy: 0.8725 - val_loss: 0.4299
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9815 - loss: 0.1363 - val_accuracy: 0.8707 - val_loss: 0.4386
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9851 - loss: 0.1278 - val_accuracy: 0.8683 - val_loss: 0.4468
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9842 - loss: 0.1285 - val_accuracy: 0.8687 - val_loss: 0.4509
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_data_1, train_data_2, epochs=8, batch_size=512)
binary_matrix_regularization = model_regularization.evaluate(test_data_1, test_data_2)
binary_matrix_regularization
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9363 - loss: 0.2650
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9465 - loss: 0.2107
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9539 - loss: 0.1921
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9536 - loss: 0.1840
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.9562 - loss: 0.1810
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9624 - loss: 0.1687
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9624 - loss: 0.1710
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.9632 - loss: 0.1649
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8649 - loss: 0.4392
[0.4368860423564911, 0.8670799732208252]
```

The loss on test set is 0.4813 and accuracy is 85.84%.

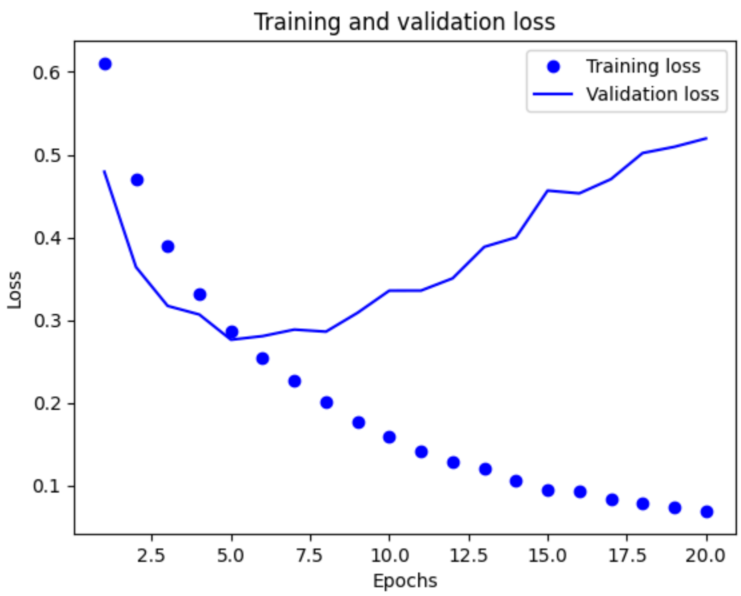
Dropout

```
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_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_Dropout = history_model_Dropout.history
history_dict_Dropout.keys()
```

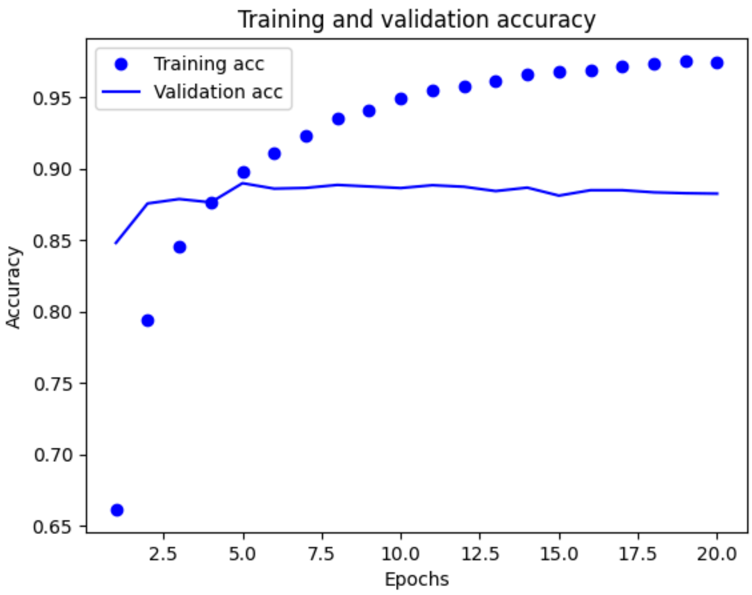
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.6052 - loss: 0.6513 - val_accuracy: 0.8481 - val_loss: 0.4793
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.7839 - loss: 0.4870 - val_accuracy: 0.8757 - val_loss: 0.3640
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.8374 - loss: 0.4026 - val_accuracy: 0.8788 - val_loss: 0.3170
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.8745 - loss: 0.3412 - val_accuracy: 0.8765 - val_loss: 0.3066
```

```
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.8947 - loss: 0.2956 - val_accuracy: 0.8899 - val_loss: 0.2761
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.9099 - loss: 0.2592 - val_accuracy: 0.8861 - val_loss: 0.2805
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9215 - loss: 0.2314 - val_accuracy: 0.8866 - val_loss: 0.2884
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9370 - loss: 0.1978 - val_accuracy: 0.8887 - val_loss: 0.2859
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9405 - loss: 0.1769 - val_accuracy: 0.8876 - val_loss: 0.3088
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9488 - loss: 0.1603 - val_accuracy: 0.8865 - val_loss: 0.3355
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9535 - loss: 0.1418 - val_accuracy: 0.8885 - val_loss: 0.3355
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9584 - loss: 0.1280 - val_accuracy: 0.8874 - val_loss: 0.3504
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9628 - loss: 0.1184 - val_accuracy: 0.8844 - val_loss: 0.3884
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9681 - loss: 0.1016 - val_accuracy: 0.8868 - val_loss: 0.3999
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9717 - loss: 0.0887 - val_accuracy: 0.8812 - val_loss: 0.4565
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.9684 - loss: 0.0969 - val_accuracy: 0.8850 - val_loss: 0.4532
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9720 - loss: 0.0845 - val_accuracy: 0.8850 - val_loss: 0.4703
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9750 - loss: 0.0775 - val_accuracy: 0.8835 - val_loss: 0.5019
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9756 - loss: 0.0747 - val_accuracy: 0.8829 - val_loss: 0.5094
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9741 - loss: 0.0710 - val_accuracy: 0.8826 - val_loss: 0.5195
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_data_1, train_data_2, epochs=8, batch_size=512)
binary_matrix_Dropout = model_Dropout.evaluate(test_data_1, test_data_2)
binary_matrix_Dropout
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9256 - loss: 0.2722
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.9350 - loss: 0.1962
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.9431 - loss: 0.1774
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.9460 - loss: 0.1577
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9491 - loss: 0.1461
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9523 - loss: 0.1359
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9555 - loss: 0.1333
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9546 - loss: 0.1267
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8707 - loss: 0.5042
[0.4997107684612274, 0.8728399872779846]
```

The loss on the test set is 0.614 and accuracy is 87%.

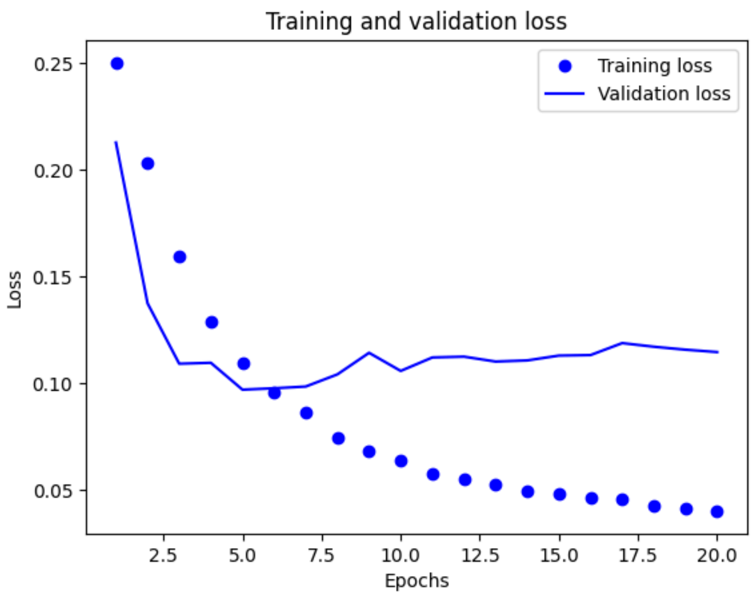
Training model with hyper tuned parameters

```
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_data_1,
                    partial_train_data_2,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
history_dict_Hyper = history_model_Hyper.history
history_dict_Hyper.keys()
```

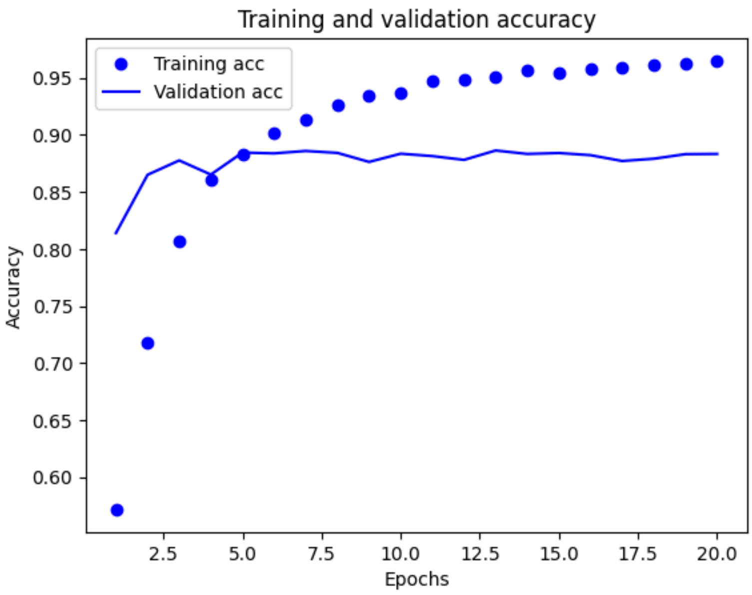
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 88ms/step - accuracy: 0.5301 - loss: 0.2587 - val_accuracy: 0.8140 - val_loss: 0.2128
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 78ms/step - accuracy: 0.6950 - loss: 0.2137 - val_accuracy: 0.8651 - val_loss: 0.1375
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.7913 - loss: 0.1666 - val_accuracy: 0.8777 - val_loss: 0.1092
```

```
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 58ms/step - accuracy: 0.8581 - loss: 0.1314 - val_accuracy: 0.8653 - val_loss: 0.1096
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.8771 - loss: 0.1130 - val_accuracy: 0.8846 - val_loss: 0.0970
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.9015 - loss: 0.0957 - val_accuracy: 0.8839 - val_loss: 0.0977
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 76ms/step - accuracy: 0.9140 - loss: 0.0872 - val_accuracy: 0.8860 - val_loss: 0.0985
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9275 - loss: 0.0738 - val_accuracy: 0.8843 - val_loss: 0.1042
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.9359 - loss: 0.0674 - val_accuracy: 0.8764 - val_loss: 0.1143
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9379 - loss: 0.0627 - val_accuracy: 0.8836 - val_loss: 0.1058
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.9466 - loss: 0.0572 - val_accuracy: 0.8815 - val_loss: 0.1121
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 0.9476 - loss: 0.0552 - val_accuracy: 0.8782 - val_loss: 0.1125
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - accuracy: 0.9508 - loss: 0.0532 - val_accuracy: 0.8864 - val_loss: 0.1102
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9567 - loss: 0.0494 - val_accuracy: 0.8834 - val_loss: 0.1107
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 48ms/step - accuracy: 0.9547 - loss: 0.0482 - val_accuracy: 0.8842 - val_loss: 0.1130
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.9587 - loss: 0.0455 - val_accuracy: 0.8823 - val_loss: 0.1132
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.9589 - loss: 0.0446 - val_accuracy: 0.8772 - val_loss: 0.1188
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 100ms/step - accuracy: 0.9635 - loss: 0.0421 - val_accuracy: 0.8792 - val_loss: 0.1171
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 81ms/step - accuracy: 0.9619 - loss: 0.0416 - val_accuracy: 0.8832 - val_loss: 0.1158
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.9640 - loss: 0.0410 - val_accuracy: 0.8834 - val_loss: 0.1146
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_data_1, train_data_2, epochs=8, batch_size=512)
binary_matrix_Hyper = model_Hyper.evaluate(test_data_1, test_data_2)
binary_matrix_Hyper
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9228 - loss: 0.0749
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 36ms/step - accuracy: 0.9322 - loss: 0.0683
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 0.9430 - loss: 0.0625
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 36ms/step - accuracy: 0.9454 - loss: 0.0593
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9478 - loss: 0.0569
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 36ms/step - accuracy: 0.9488 - loss: 0.0566
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9527 - loss: 0.0537
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 56ms/step - accuracy: 0.9546 - loss: 0.0503
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8803 - loss: 0.1144
[0.11389057338237762, 0.8805599808692932]
```

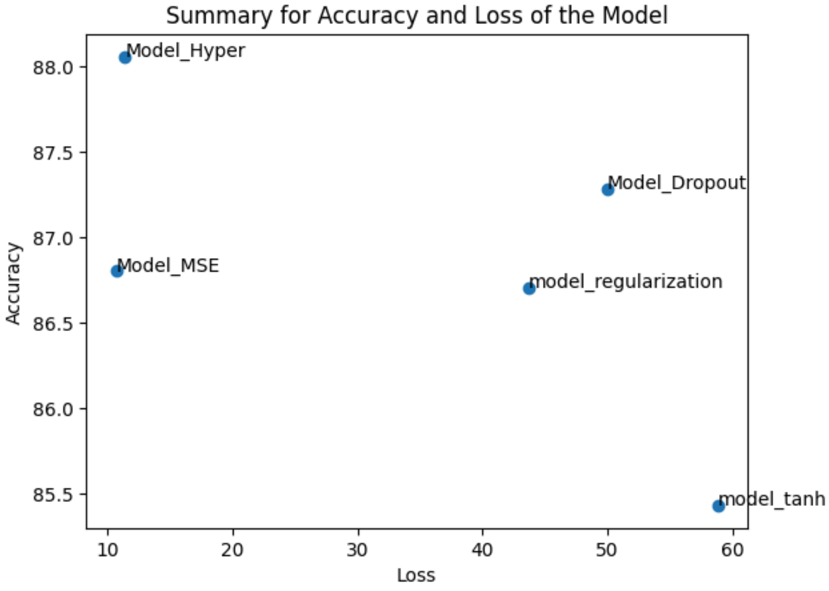
Summary

```
All_Models_Loss= np.array([binary_matrix_Dropout[0],binary_matrix_Hyper[0],binary_matrix_MSE[0],binary_matrix_regularizatior
All_Models_Loss
All_Models_Accuracy= np.array([binary_matrix_Dropout[1],binary_matrix_Hyper[1],binary_matrix_MSE[1],binary_matrix_regulariza
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.