Summary: Some models underperformed compared to others, but there was not a model that stood definitively above all of the others.

The complex models (2+ representation layers, over 16 units per layer) seemed to perform best on the training data, but not on the test data. For this data set, the simpler models (1 layer, 8 units per layer) were a bit more robust, and often performed better on the test data.

I found using 1 representation layer with 16 units to be the best choice.

I tried several additional models to see if any trends emerged.

Other observations included the relatively poor performance of the tanh activation function, and the very low loss value of the MSE loss function, even with the mediocre accuracy it provided. Also, using the dropout regularization technique tended to result it better validation accuracy but I was not able to get better results overall on test data.

```
import pandas as pd

# Summary table
data = {
    "Layers (representation only)": [1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1],
    "Units": [16, 32, 16, 16, 16, 4, 8, 32, 16, 8, 4],
    "Special": ["-","-","MSE loss","tanh","dropout", "dropout",
"dropout", "dropout", "dropout", "dropout", "dropout"],
    "Epochs": [5, 3, 5, 3, 6, 3, 7, 6, 7, 7, 12],
    "Loss": [0.279, 0.317, 0.106, 0.460, 0.446, 0.472, 0.367, 0.473,
0.328, 0.289, 0.321],
    "Accuracy": [0.888, 0.871, 0.867, 0.858, 0.872, 0.881, 0.875,
0.873, 0.879, 0.883, 0.878]

}

# Create a data frame
df = pd.DataFrame(data)

# Display the data frame
print(df)
```

```
    Layers (representation only)  Units   Special  Epochs   Loss
Accuracy
0                              1     16         -       5  0.279
0.888
1                              2     32         -       3  0.317
0.871
2                              2     16  MSE loss       5  0.106
0.867
3                              2     16      tanh       3  0.460
0.858
4                              2     16   dropout       6  0.446
```

```
0.872
5                              2       4    dropout        3  0.472
0.881
6                              2       8    dropout        7  0.367
0.875
7                              2      32    dropout        6  0.473
0.873
8                              1      16    dropout        7  0.328
0.879
9                              1       8    dropout        7  0.289
0.883
10                             1       4    dropout       12  0.321
0.878
```

1. Here is the IMDB classification task using just **one representation layer**, instead of two.

```
from tensorflow.keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)

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

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

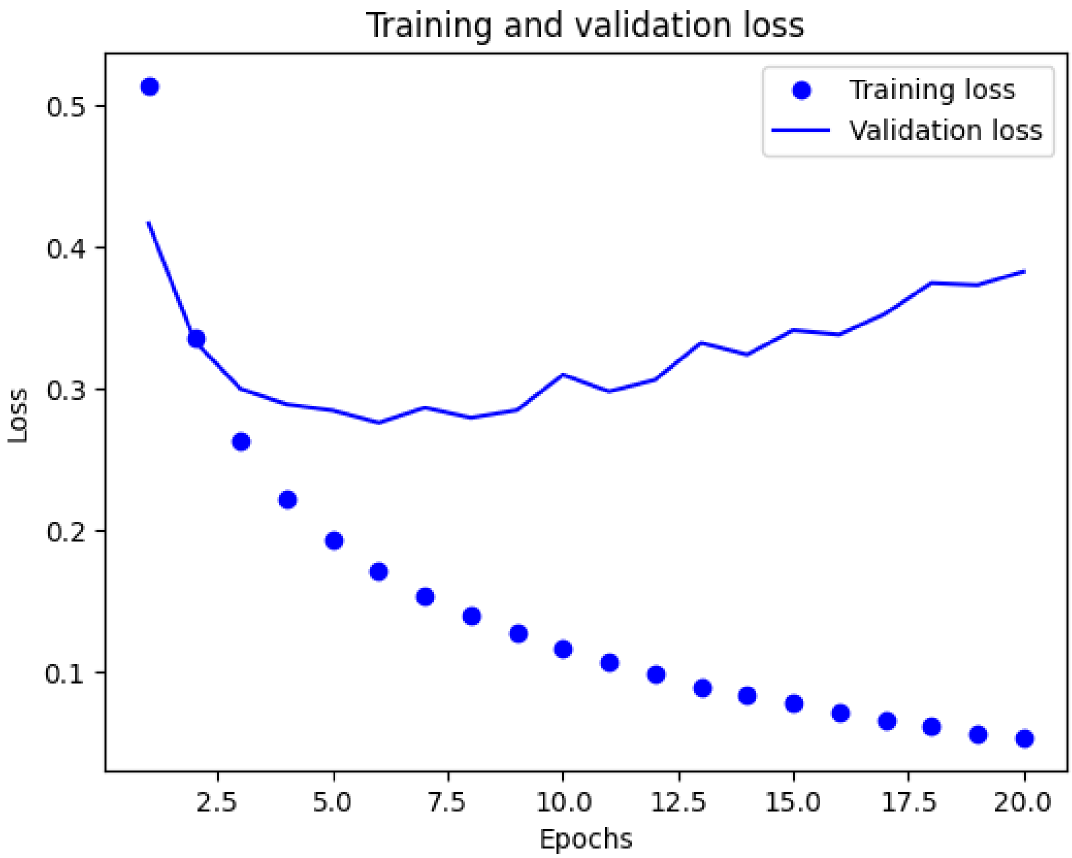
The model will be built with just one representation layer, and one final classification layer.

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

# Compiling the model
model.compile(optimizer="rmsprop",
```

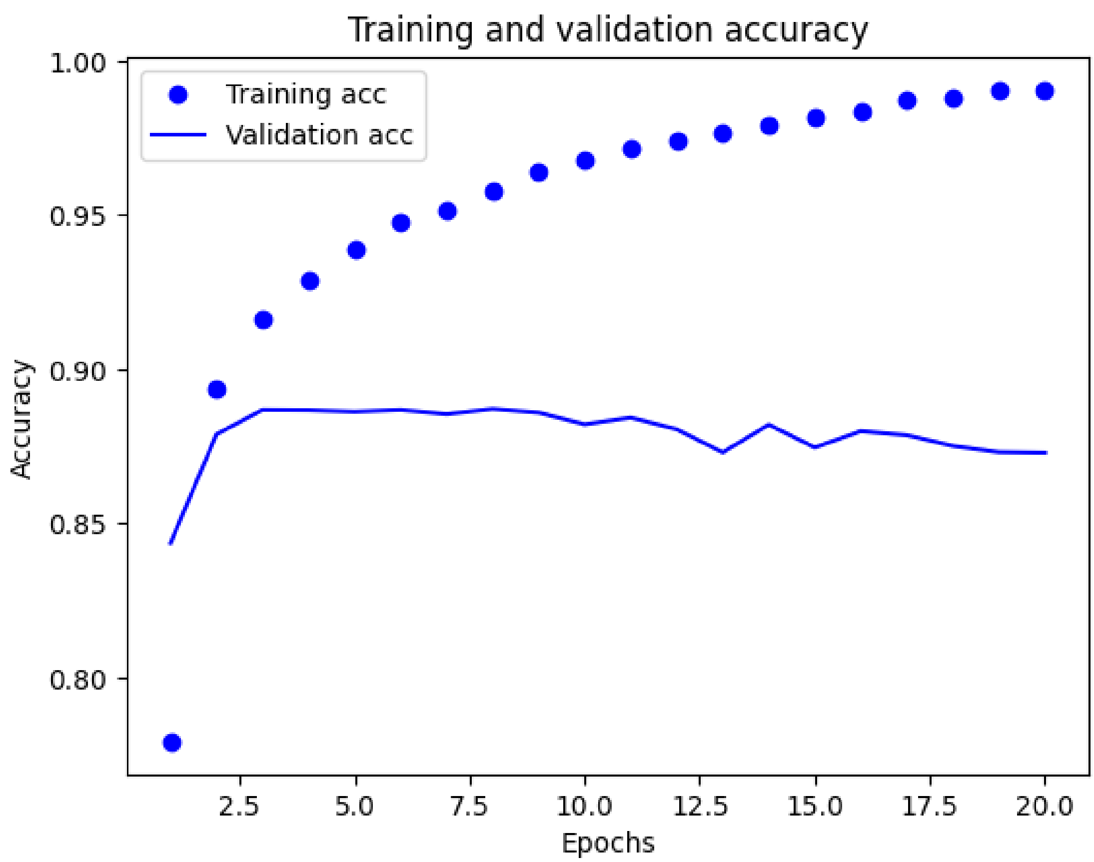
```python
             loss="binary_crossentropy",
             metrics=["accuracy"])

# Splitting the training and validation sets
x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - accuracy: 0.6949 - loss:
0.5866 - val_accuracy: 0.8436 - val_loss: 0.4163
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.8884 - loss:
0.3534 - val_accuracy: 0.8789 - val_loss: 0.3333
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9224 - loss:
0.2618 - val_accuracy: 0.8868 - val_loss: 0.2996
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9299 - loss:
0.2259 - val_accuracy: 0.8867 - val_loss: 0.2888
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9431 - loss:
0.1902 - val_accuracy: 0.8862 - val_loss: 0.2844
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9493 - loss:
0.1745 - val_accuracy: 0.8868 - val_loss: 0.2758
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9532 - loss:
0.1509 - val_accuracy: 0.8854 - val_loss: 0.2864
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.9615 - loss:
0.1355 - val_accuracy: 0.8871 - val_loss: 0.2793
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9646 - loss:
0.1279 - val_accuracy: 0.8859 - val_loss: 0.2846
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9695 - loss:
0.1151 - val_accuracy: 0.8821 - val_loss: 0.3097
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9717 - loss:
0.1043 - val_accuracy: 0.8843 - val_loss: 0.2978
Epoch 12/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9757 - loss:
0.0966 - val_accuracy: 0.8805 - val_loss: 0.3061
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9764 - loss:
0.0904 - val_accuracy: 0.8730 - val_loss: 0.3321
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9777 - loss:
0.0876 - val_accuracy: 0.8820 - val_loss: 0.3238
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9831 - loss:
0.0758 - val_accuracy: 0.8746 - val_loss: 0.3410
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9841 - loss:
0.0732 - val_accuracy: 0.8799 - val_loss: 0.3379
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 0.9885 - loss:
0.0654 - val_accuracy: 0.8786 - val_loss: 0.3526
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9894 - loss:
0.0594 - val_accuracy: 0.8751 - val_loss: 0.3742
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9900 - loss:
0.0575 - val_accuracy: 0.8731 - val_loss: 0.3728
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9910 - loss:
0.0521 - val_accuracy: 0.8729 - val_loss: 0.3823

# 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()
```




```
#  Plotting the training and validation accuracy
plt.clf()
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



The validation loss and accuracy appear to optimize at approximately the **5th epoch**. The model can be retrained for 5 epochs, one activation layer, and one classification layer, then evaluated on the test data:

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=5, batch_size=512)
results = model.evaluate(x_test, y_test)

Epoch 1/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.7435 - loss:
0.5510
Epoch 2/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.9016 - loss:
0.3071
Epoch 3/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9182 - loss:
0.2408
```

```
Epoch 4/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9330 - loss: 
0.2049
Epoch 5/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.9385 - loss: 
0.1854
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8863 - loss: 
0.2806
```

```
results
```

```
[0.27903759479522705, 0.8882399797439575]
```

The final results for using one representation layer and 5 epochs are an **accuracy of 0.882 and a loss value of 0.279.**

---

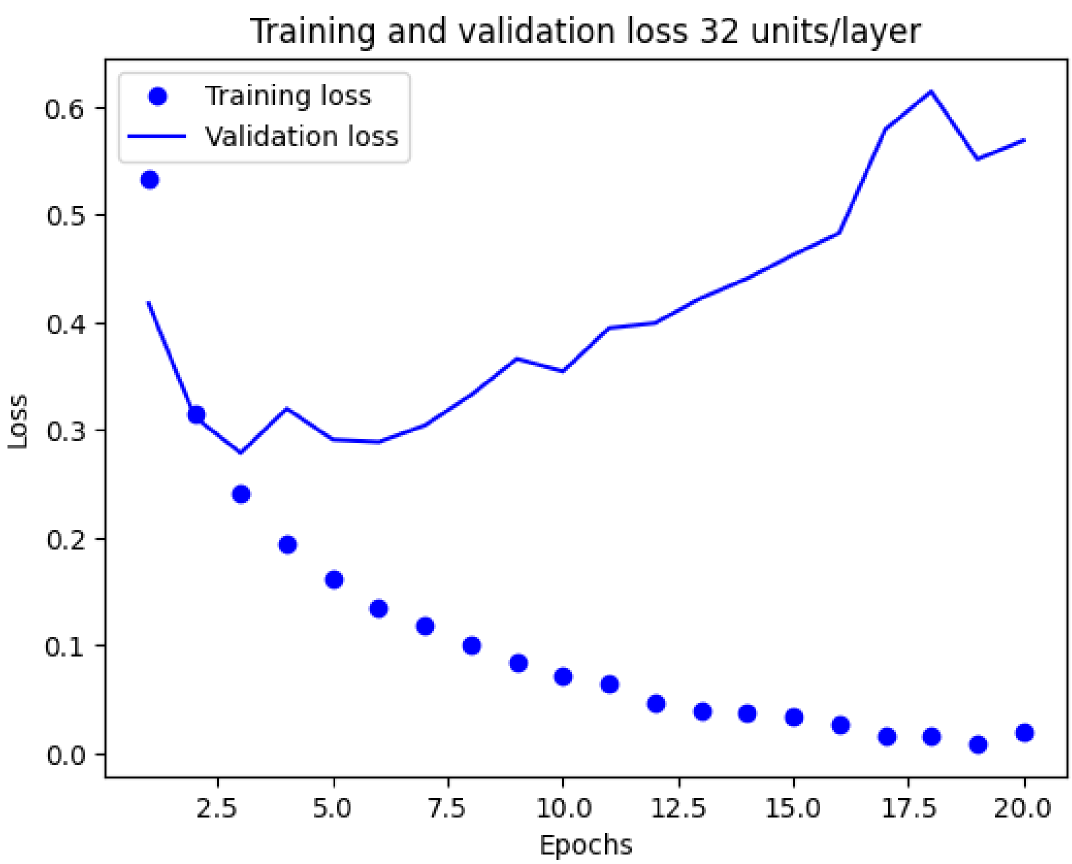
2. Here is the IMDB classification task using **32 units in each representation layer**, instead of **16**. Two representation layers and one classification layer will be used.

```
# Building the model
model = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

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

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

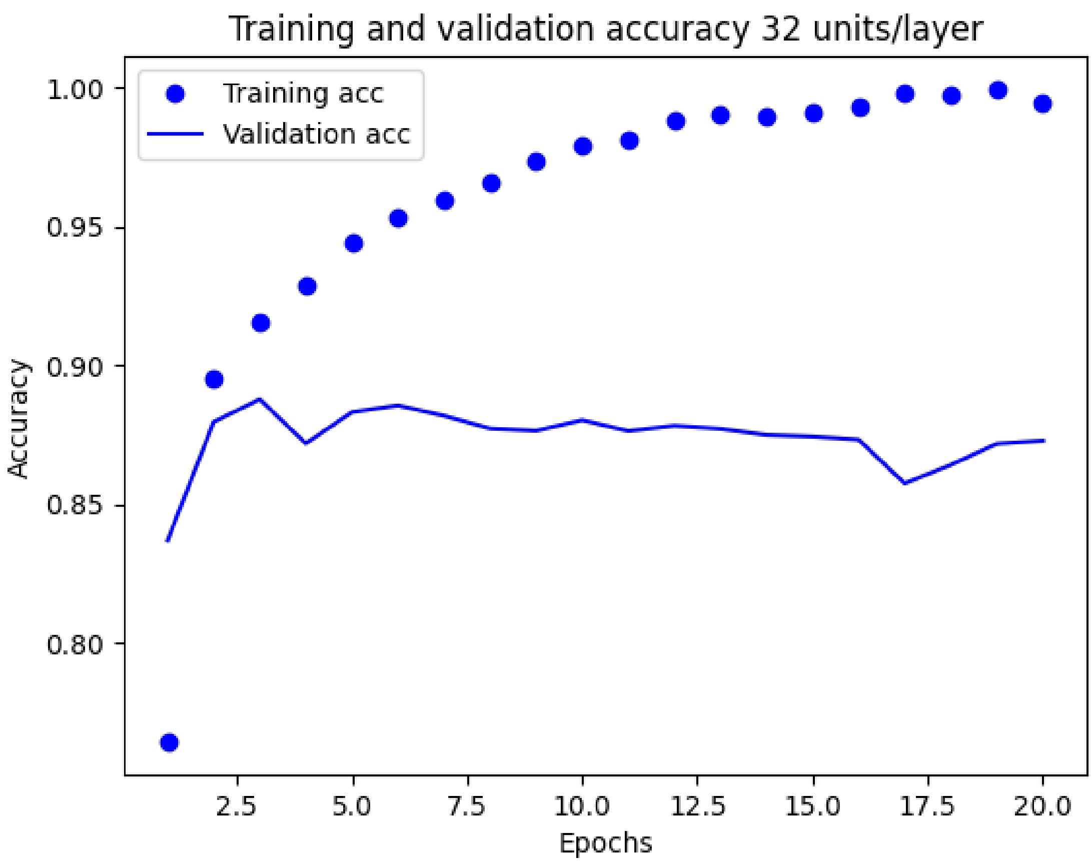
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 91ms/step - accuracy: 0.6793 - loss: 
0.6051 - val_accuracy: 0.8369 - val_loss: 0.4174
Epoch 2/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 46ms/step - accuracy: 0.8904 - loss:
0.3388 - val_accuracy: 0.8796 - val_loss: 0.3121
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9121 - loss:
0.2489 - val_accuracy: 0.8877 - val_loss: 0.2786
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 54ms/step - accuracy: 0.9359 - loss:
0.1905 - val_accuracy: 0.8718 - val_loss: 0.3198
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.9427 - loss:
0.1630 - val_accuracy: 0.8831 - val_loss: 0.2910
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 50ms/step - accuracy: 0.9552 - loss:
0.1348 - val_accuracy: 0.8855 - val_loss: 0.2889
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9644 - loss:
0.1125 - val_accuracy: 0.8818 - val_loss: 0.3042
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - accuracy: 0.9710 - loss:
0.0929 - val_accuracy: 0.8772 - val_loss: 0.3323
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - accuracy: 0.9789 - loss:
0.0780 - val_accuracy: 0.8764 - val_loss: 0.3660
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9817 - loss:
0.0669 - val_accuracy: 0.8802 - val_loss: 0.3543
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9872 - loss:
0.0540 - val_accuracy: 0.8763 - val_loss: 0.3946
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 52ms/step - accuracy: 0.9915 - loss:
0.0413 - val_accuracy: 0.8782 - val_loss: 0.3993
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9925 - loss:
0.0363 - val_accuracy: 0.8771 - val_loss: 0.4225
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - accuracy: 0.9924 - loss:
0.0318 - val_accuracy: 0.8749 - val_loss: 0.4406
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9943 - loss:
0.0268 - val_accuracy: 0.8743 - val_loss: 0.4625
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9974 - loss:
0.0191 - val_accuracy: 0.8732 - val_loss: 0.4827
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - accuracy: 0.9995 - loss:
0.0130 - val_accuracy: 0.8574 - val_loss: 0.5790
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9988 - loss:
```

```
0.0141 - val_accuracy: 0.8641 - val_loss: 0.6143
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9991 - loss:
0.0108 - val_accuracy: 0.8717 - val_loss: 0.5514
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9965 - loss:
0.0145 - val_accuracy: 0.8728 - val_loss: 0.5688

# Plotting the training and validation loss
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 32 units/layer')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
#  Plotting the training and validation accuracy
plt.clf()
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy 32 units/layer')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



The validation loss and accuracy appear to optimize at approximately the **3rd epoch**. The model can be retrained for 3 epochs, 32 units, two activation layers, and one classification layer, then evaluated on the test data:

```
model = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
```

```
                loss="binary_crossentropy",
                metrics=["accuracy"])
model.fit(x_train, y_train, epochs=3, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.7207 - loss:
0.5504
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 55ms/step - accuracy: 0.9019 - loss:
0.2715
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 33ms/step - accuracy: 0.9261 - loss:
0.2063
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8709 - loss:
0.3148
```

```
results
```

```
[0.31669163703918457, 0.8712400197982788]
```

The final results for using 32 units, two representation layers and 3 epochs are an **accuracy of 0.871 and a loss value of 0.317.**

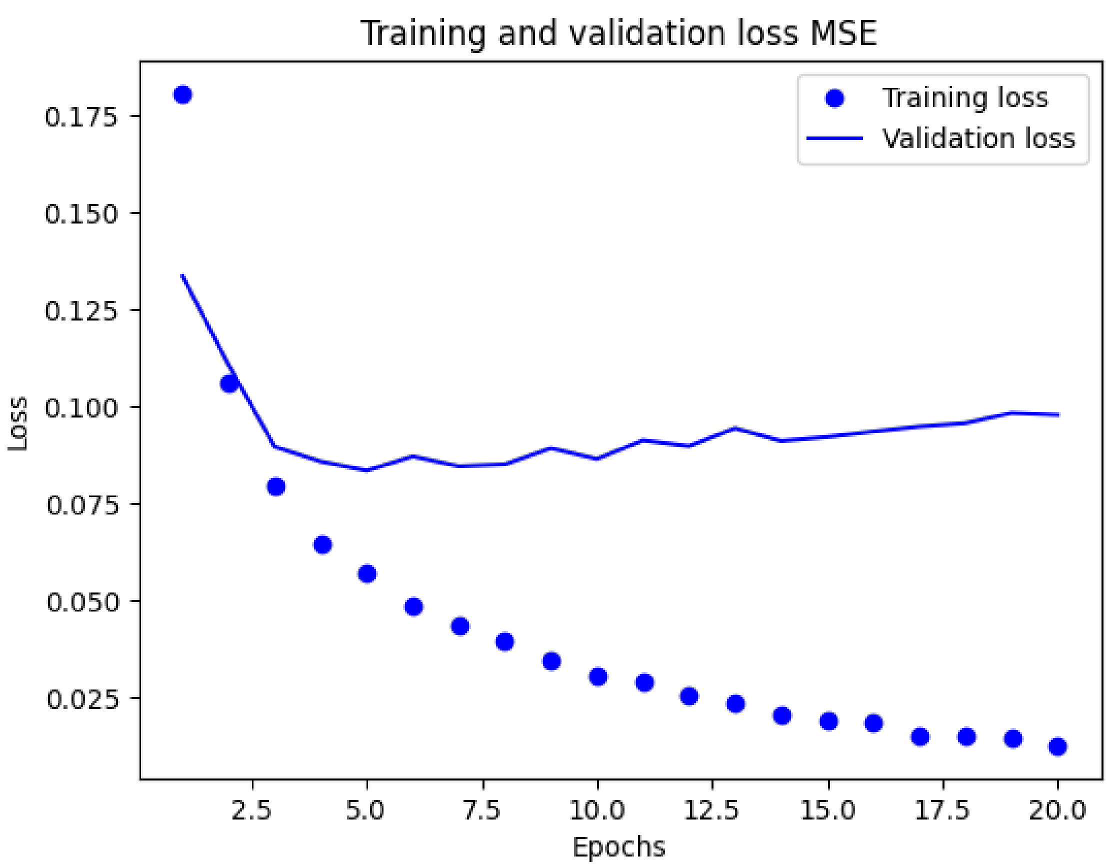
---

3. Now the **MSE** loss function will be applied in place of the binary_crossentropy loss function.

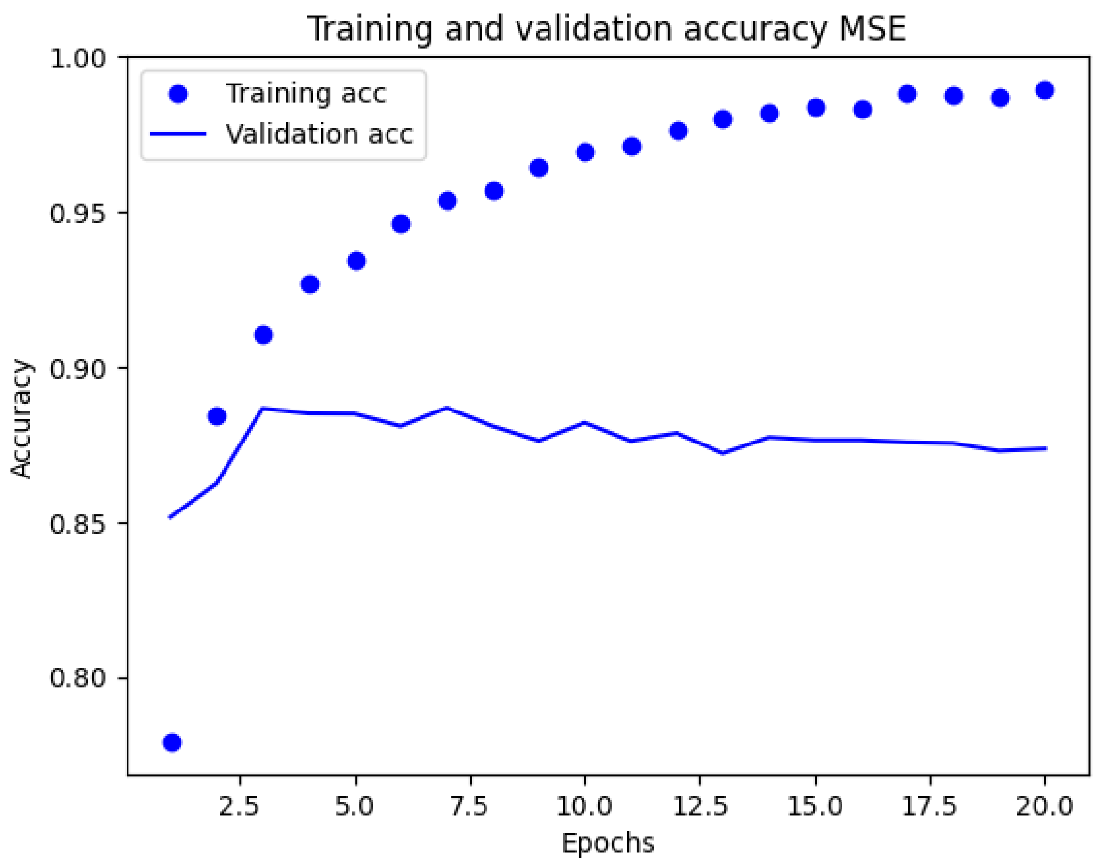
```
# Building the model
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

# Compiling the model with the MSE loss function
model.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.7016 - loss: 
0.2107 - val_accuracy: 0.8516 - val_loss: 0.1335
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.8772 - loss: 
0.1142 - val_accuracy: 0.8624 - val_loss: 0.1106
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9124 - loss: 
0.0816 - val_accuracy: 0.8866 - val_loss: 0.0896
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9273 - loss: 
0.0652 - val_accuracy: 0.8850 - val_loss: 0.0858
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9349 - loss: 
0.0574 - val_accuracy: 0.8849 - val_loss: 0.0835
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9498 - loss: 
0.0475 - val_accuracy: 0.8809 - val_loss: 0.0871
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9546 - loss: 
0.0429 - val_accuracy: 0.8868 - val_loss: 0.0846
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9600 - loss: 
0.0383 - val_accuracy: 0.8809 - val_loss: 0.0851
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9689 - loss: 
0.0320 - val_accuracy: 0.8762 - val_loss: 0.0892
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9729 - loss: 
0.0295 - val_accuracy: 0.8820 - val_loss: 0.0865
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9746 - loss: 
0.0285 - val_accuracy: 0.8761 - val_loss: 0.0913
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9778 - loss: 
0.0247 - val_accuracy: 0.8787 - val_loss: 0.0898
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9825 - loss: 
0.0219 - val_accuracy: 0.8721 - val_loss: 0.0944
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9848 - loss: 
0.0197 - val_accuracy: 0.8773 - val_loss: 0.0911
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9873 - loss: 
0.0174 - val_accuracy: 0.8764 - val_loss: 0.0922
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9874 - loss: 
0.0163 - val_accuracy: 0.8764 - val_loss: 0.0935
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9897 - loss: 
```

```
0.0142 - val_accuracy: 0.8757 - val_loss: 0.0948
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - accuracy: 0.9891 - loss:
0.0143 - val_accuracy: 0.8754 - val_loss: 0.0957
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9882 - loss:
0.0138 - val_accuracy: 0.8729 - val_loss: 0.0983
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9894 - loss:
0.0130 - val_accuracy: 0.8736 - val_loss: 0.0978

# Plotting the training and validation loss
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 MSE')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
#  Plotting the training and validation accuracy
plt.clf()
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy MSE')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



The validation loss and accuracy appear to optimize at approximately the 5th epoch using the MSE loss function. The model can be retrained for 5 epochs, two activation layers, 16 units, and one classification layer, then evaluated on the test data:

```
model.fit(x_train, y_train, epochs=5, batch_size=512)
results = model.evaluate(x_test, y_test)
results

Epoch 1/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9446 - loss: 
0.0470
```

```
Epoch 2/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9593 - loss:
0.0374
Epoch 3/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9648 - loss:
0.0321
Epoch 4/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9704 - loss:
0.0283
Epoch 5/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.9713 - loss:
0.0278
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8628 - loss:
0.1082

[0.10566967725753784, 0.8663399717330933]
```

Using the MSE loss function with two activation layers of 16 units each with 6 epochs shows an accuracy value of .867 and a loss value of .106. While MSE does not show the highest accuracy compared to other loss functions, it is remarkable how low the loss value is compared to other loss functions.

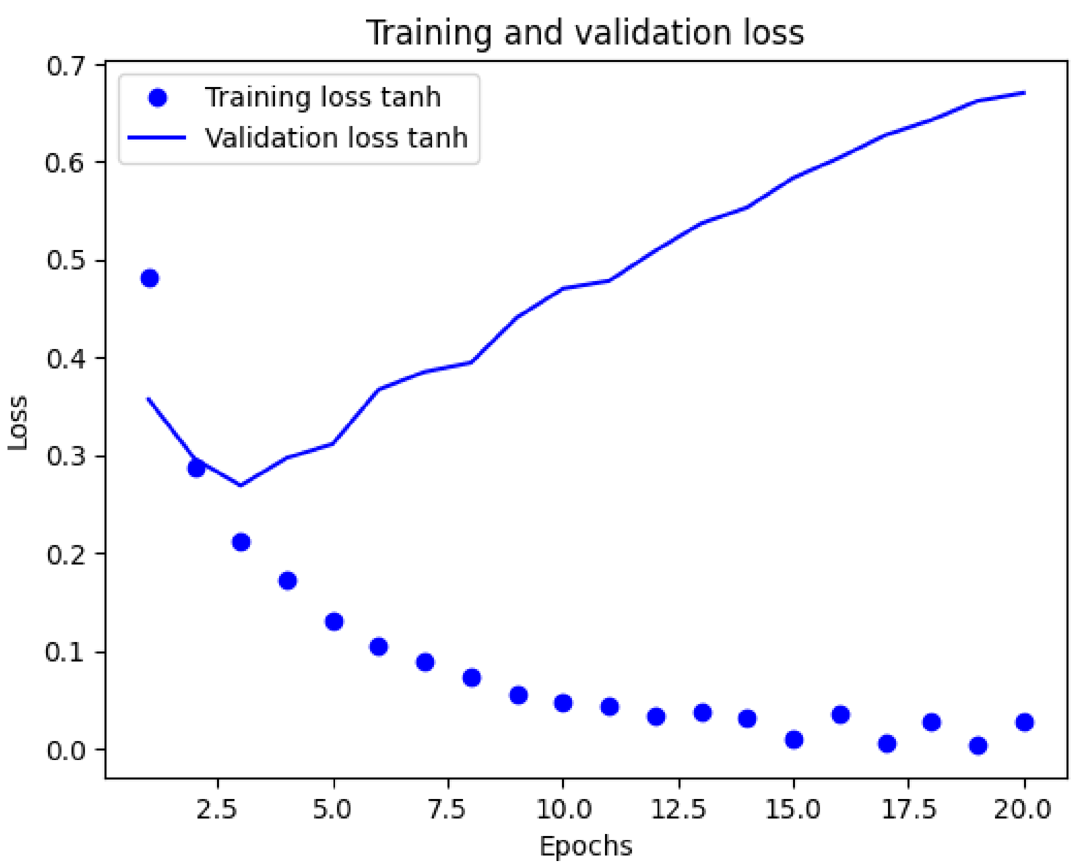
---

4. Now the **tanh** activation method will be applied in place of relu.

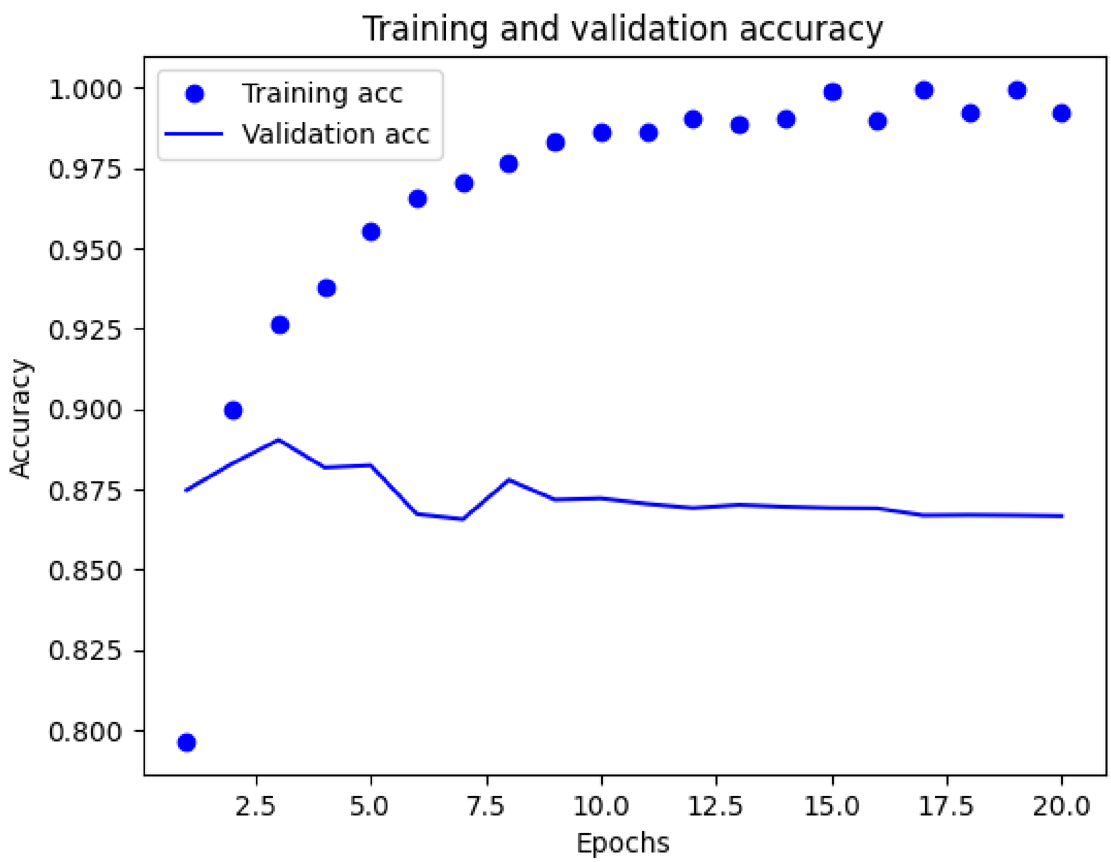
```
# Building the model
model = keras.Sequential([
    layers.Dense(16, activation="tanh"),
    layers.Dense(16, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])

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

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))

Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 126ms/step - accuracy: 0.7276 - loss:
```

```
0.5585 - val_accuracy: 0.8747 - val_loss: 0.3570
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 90ms/step - accuracy: 0.9057 - loss:
0.2968 - val_accuracy: 0.8830 - val_loss: 0.2962
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 49ms/step - accuracy: 0.9293 - loss:
0.2145 - val_accuracy: 0.8903 - val_loss: 0.2691
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9427 - loss:
0.1683 - val_accuracy: 0.8817 - val_loss: 0.2971
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9598 - loss:
0.1249 - val_accuracy: 0.8824 - val_loss: 0.3114
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9691 - loss:
0.1005 - val_accuracy: 0.8672 - val_loss: 0.3672
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9685 - loss:
0.0928 - val_accuracy: 0.8656 - val_loss: 0.3852
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9817 - loss:
0.0647 - val_accuracy: 0.8778 - val_loss: 0.3944
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9846 - loss:
0.0526 - val_accuracy: 0.8717 - val_loss: 0.4407
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9862 - loss:
0.0482 - val_accuracy: 0.8721 - val_loss: 0.4702
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9894 - loss:
0.0396 - val_accuracy: 0.8704 - val_loss: 0.4780
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9929 - loss:
0.0286 - val_accuracy: 0.8691 - val_loss: 0.5087
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9923 - loss:
0.0279 - val_accuracy: 0.8701 - val_loss: 0.5367
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9962 - loss:
0.0184 - val_accuracy: 0.8695 - val_loss: 0.5531
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9990 - loss:
0.0102 - val_accuracy: 0.8691 - val_loss: 0.5832
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9930 - loss:
0.0253 - val_accuracy: 0.8690 - val_loss: 0.6038
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9997 - loss:
0.0054 - val_accuracy: 0.8668 - val_loss: 0.6269
```

```
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9951 - loss:
0.0197 - val_accuracy: 0.8669 - val_loss: 0.6424
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9998 - loss:
0.0034 - val_accuracy: 0.8668 - val_loss: 0.6619
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9915 - loss:
0.0288 - val_accuracy: 0.8666 - val_loss: 0.6702

# Plotting the training and validation loss
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 tanh')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss tanh')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
#  Plotting the training and validation accuracy
plt.clf()
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Using the tanh activation function, the accuracy and loss is optimized after 3 epochs on the validation set.

```
model.fit(x_train, y_train, epochs=3, batch_size=512)
results = model.evaluate(x_test, y_test)
results

Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9438 - loss:
0.2639
Epoch 2/3
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9639 - loss:
0.1260
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.9648 - loss:
0.1101
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8554 - loss:
0.4686

[0.4603240489959717, 0.8580399751663208]
```

The tanh activation function performed worse than the relu function, with an accuracy of .858 and a much larger loss of 0.460. The tanh activation function appears to be overfitting the training data; it shows very high accuracy and low loss values on the training data, but the results on the test data are not as high as other activation functions.

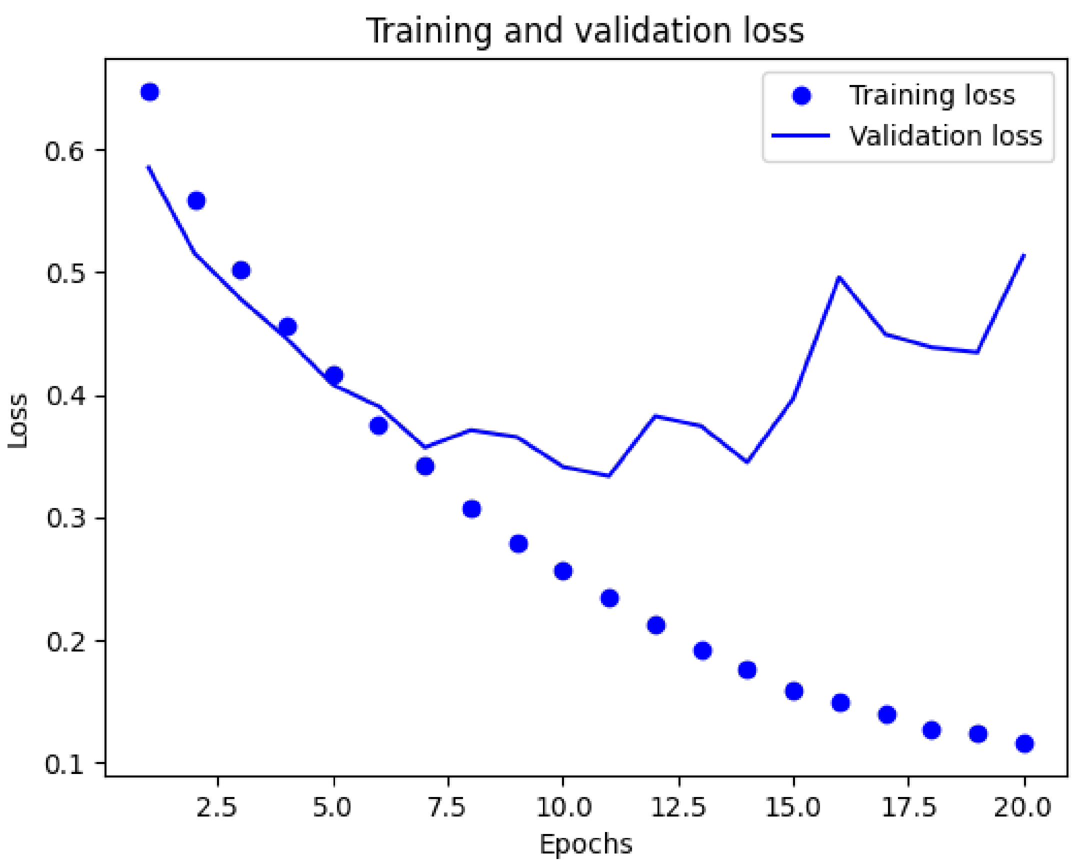
---

5. Now the **dropout** regularization technique will be applied, using 16 units, 2 representation layers, and the rmsprop loss function.

```
# building the model
model = 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")
])

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

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))

Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.5717 - loss:
0.6730 - val_accuracy: 0.6202 - val_loss: 0.5851
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.7622 - loss:
```

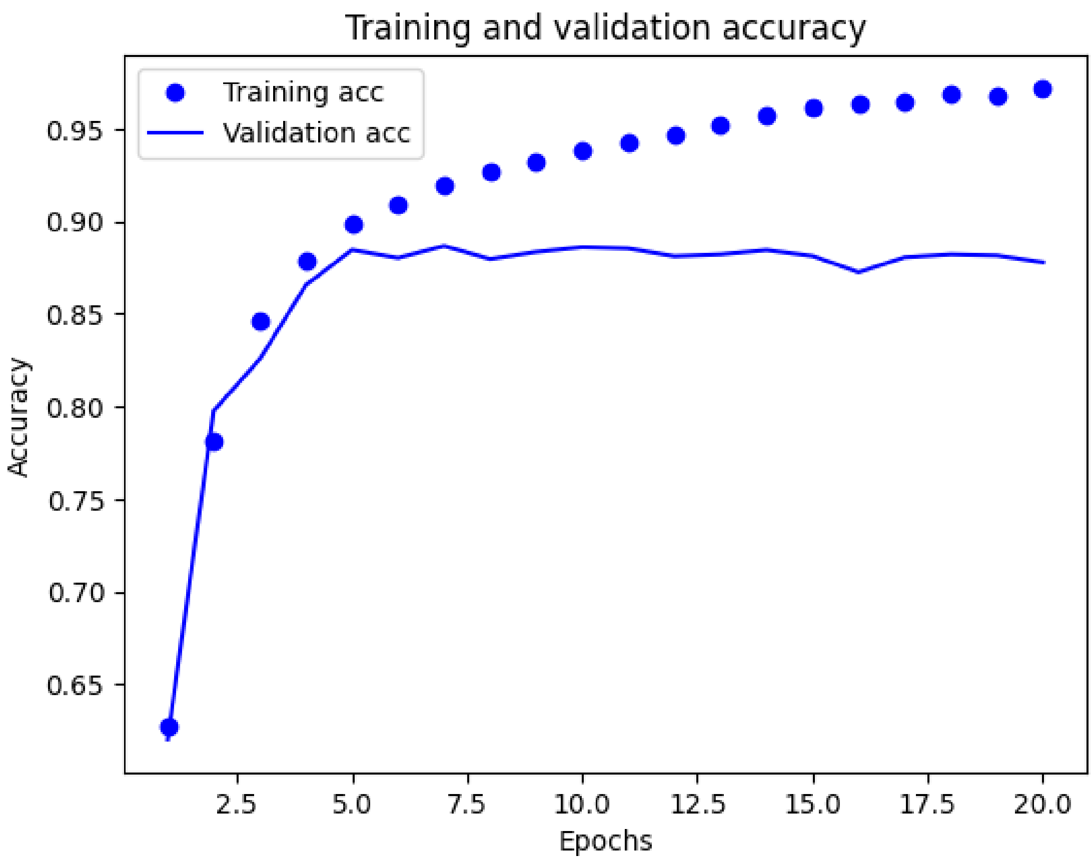
```
0.5685 - val_accuracy: 0.7977 - val_loss: 0.5152
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 66ms/step - accuracy: 0.8405 - loss:
0.5087 - val_accuracy: 0.8255 - val_loss: 0.4780
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.8776 - loss:
0.4608 - val_accuracy: 0.8655 - val_loss: 0.4457
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.8995 - loss:
0.4263 - val_accuracy: 0.8845 - val_loss: 0.4082
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9093 - loss:
0.3820 - val_accuracy: 0.8802 - val_loss: 0.3904
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9181 - loss:
0.3450 - val_accuracy: 0.8866 - val_loss: 0.3570
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9265 - loss:
0.3108 - val_accuracy: 0.8796 - val_loss: 0.3711
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9329 - loss:
0.2799 - val_accuracy: 0.8834 - val_loss: 0.3654
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9364 - loss:
0.2623 - val_accuracy: 0.8860 - val_loss: 0.3413
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9428 - loss:
0.2357 - val_accuracy: 0.8853 - val_loss: 0.3336
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.9476 - loss:
0.2115 - val_accuracy: 0.8811 - val_loss: 0.3825
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9532 - loss:
0.1926 - val_accuracy: 0.8821 - val_loss: 0.3743
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9581 - loss:
0.1767 - val_accuracy: 0.8844 - val_loss: 0.3449
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9624 - loss:
0.1568 - val_accuracy: 0.8812 - val_loss: 0.3966
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9645 - loss:
0.1513 - val_accuracy: 0.8725 - val_loss: 0.4958
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9663 - loss:
0.1394 - val_accuracy: 0.8805 - val_loss: 0.4493
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9701 - loss:
0.1279 - val_accuracy: 0.8821 - val_loss: 0.4385
```

```
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9676 - loss:
0.1243 - val_accuracy: 0.8815 - val_loss: 0.4345
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9720 - loss:
0.1177 - val_accuracy: 0.8778 - val_loss: 0.5132

# Plotting the training and validation loss
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()
```




```
#  Plotting the training and validation accuracy
plt.clf()
```

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



With dropout applied, 6 epochs shows the best values for accuracy and loss.

```
model.fit(x_train, y_train, epochs=6, batch_size=512)
results = model.evaluate(x_test, y_test)
results

Epoch 1/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9221 - loss:
0.2679
Epoch 2/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9329 - loss:
0.2294
Epoch 3/6
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9416 - loss:
0.2005
Epoch 4/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9469 - loss:
0.1850
Epoch 5/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.9481 - loss:
0.1763
Epoch 6/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9518 - loss:
0.1625
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8689 - loss:
0.4445

[0.44635409116744995, 0.872160017490387]
```

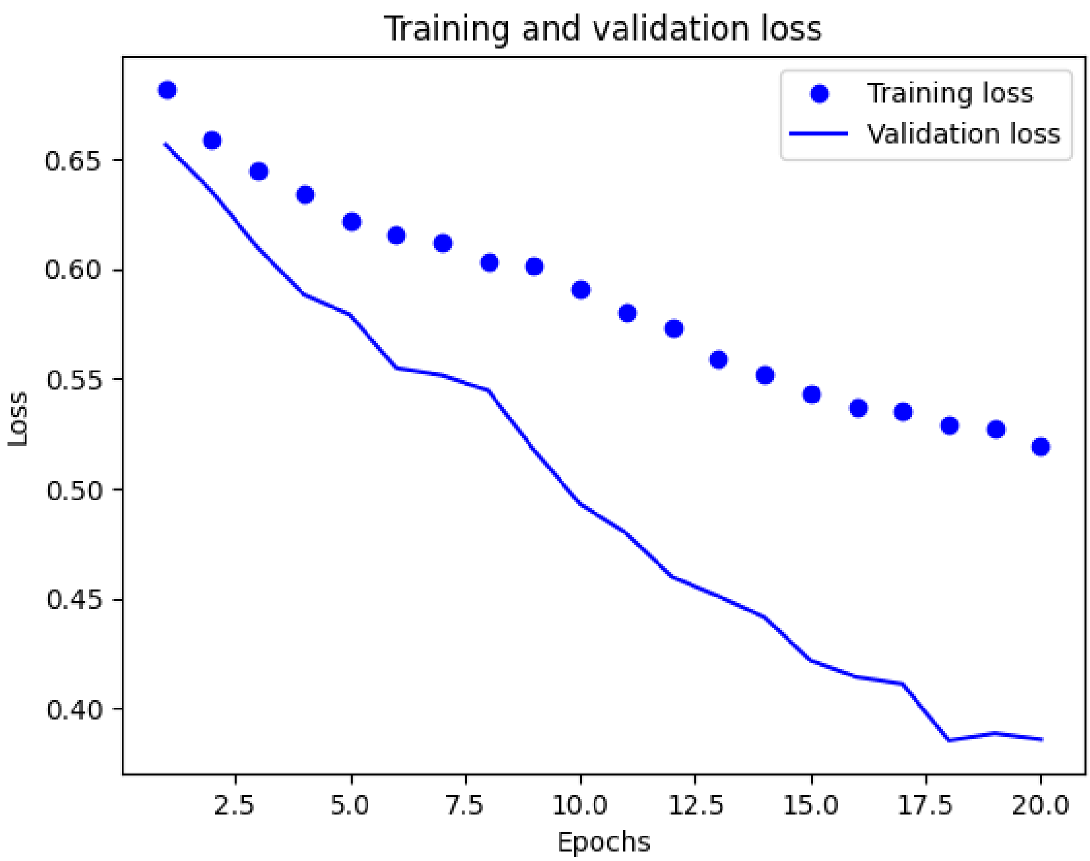
---

1. Attempting to simplify the model by reducing the units in each layer to 4.

```
# building the model
model = keras.Sequential([
    layers.Dense(4, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(4, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])

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

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))

Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 65ms/step - accuracy: 0.5252 - loss:
0.6882 - val_accuracy: 0.7873 - val_loss: 0.6566
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.5828 - loss:
0.6623 - val_accuracy: 0.8241 - val_loss: 0.6354
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.6101 - loss:
0.6464 - val_accuracy: 0.8438 - val_loss: 0.6096
Epoch 4/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.6270 - loss:
0.6353 - val_accuracy: 0.8372 - val_loss: 0.5884
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.6333 - loss:
0.6241 - val_accuracy: 0.8675 - val_loss: 0.5791
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.6365 - loss:
0.6182 - val_accuracy: 0.8656 - val_loss: 0.5549
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.6369 - loss:
0.6136 - val_accuracy: 0.8753 - val_loss: 0.5515
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.6457 - loss:
0.6060 - val_accuracy: 0.8739 - val_loss: 0.5447
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.6541 - loss:
0.6026 - val_accuracy: 0.8728 - val_loss: 0.5175
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.6380 - loss:
0.5953 - val_accuracy: 0.8649 - val_loss: 0.4929
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.6362 - loss:
0.5843 - val_accuracy: 0.8785 - val_loss: 0.4796
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.6329 - loss:
0.5750 - val_accuracy: 0.8775 - val_loss: 0.4597
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.6427 - loss:
0.5607 - val_accuracy: 0.8781 - val_loss: 0.4509
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.6511 - loss:
0.5512 - val_accuracy: 0.8791 - val_loss: 0.4414
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.6543 - loss:
0.5455 - val_accuracy: 0.8780 - val_loss: 0.4216
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.6615 - loss:
0.5347 - val_accuracy: 0.8779 - val_loss: 0.4142
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.6611 - loss:
0.5335 - val_accuracy: 0.8739 - val_loss: 0.4109
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.6594 - loss:
0.5337 - val_accuracy: 0.8782 - val_loss: 0.3852
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.6576 - loss:
0.5300 - val_accuracy: 0.8788 - val_loss: 0.3885
Epoch 20/20
```

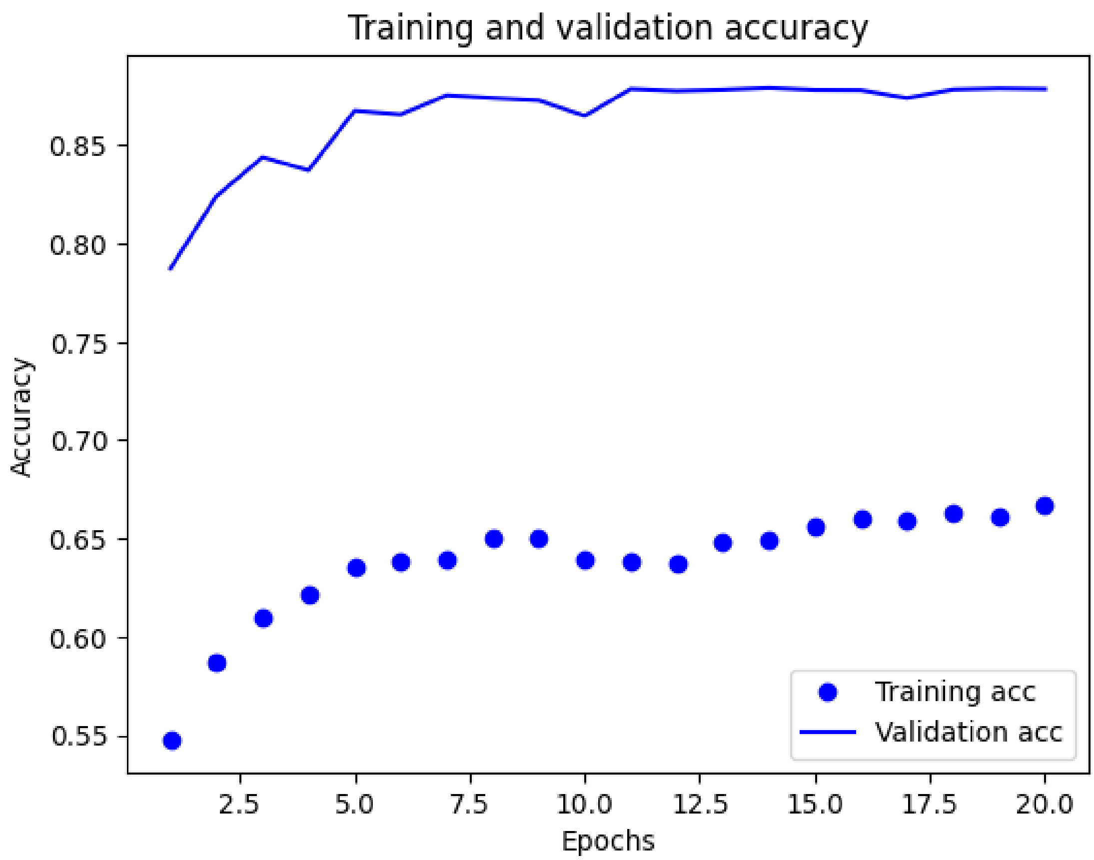
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.6732 - loss:
0.5156 - val_accuracy: 0.8785 - val_loss: 0.3857

# Plotting the training and validation loss
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()
```



Validation loss is better than training (!). As is validation accuracy below.

```
#  Plotting the training and validation accuracy
plt.clf()
acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
```

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




```
model = keras.Sequential([
    layers.Dense(4, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(4, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=3, batch_size=512)

Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.5429 - loss:
0.6735
```

```
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.6606 - loss:
0.6052
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.7113 - loss:
0.5632
```

```
<keras.src.callbacks.history.History at 0x7d7e9a291e90>
```

```
results = model.evaluate(x_test, y_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8801 - loss:
0.4729
```

```
results
```

```
[0.472202330827713, 0.8812400102615356]
```

---

another attempt with 8 units and dropout

```
# building the model
model = keras.Sequential([
    layers.Dense(8, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(8, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])

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

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=12,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 85ms/step - accuracy: 0.5586 - loss:
0.6756 - val_accuracy: 0.8032 - val_loss: 0.5751
Epoch 2/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 37ms/step - accuracy: 0.6889 - loss:
0.5831 - val_accuracy: 0.8477 - val_loss: 0.4981
Epoch 3/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.7417 - loss:
```

```
0.5247 - val_accuracy: 0.8673 - val_loss: 0.4535
Epoch 4/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.7871 - loss:
0.4802 - val_accuracy: 0.8792 - val_loss: 0.3923
Epoch 5/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.8079 - loss:
0.4435 - val_accuracy: 0.8839 - val_loss: 0.3631
Epoch 6/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.8282 - loss:
0.4116 - val_accuracy: 0.8796 - val_loss: 0.3416
Epoch 7/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8407 - loss:
0.3914 - val_accuracy: 0.8866 - val_loss: 0.3211
Epoch 8/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.8566 - loss:
0.3645 - val_accuracy: 0.8815 - val_loss: 0.3182
Epoch 9/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.8641 - loss:
0.3428 - val_accuracy: 0.8849 - val_loss: 0.2957
Epoch 10/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.8776 - loss:
0.3238 - val_accuracy: 0.8843 - val_loss: 0.3061
Epoch 11/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.8826 - loss:
0.3130 - val_accuracy: 0.8859 - val_loss: 0.2943
Epoch 12/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.8899 - loss:
0.2999 - val_accuracy: 0.8867 - val_loss: 0.2939

model.fit(x_train, y_train, epochs=7, batch_size=512)
results = model.evaluate(x_test, y_test)

Epoch 1/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8671 - loss:
0.3490
Epoch 2/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8763 - loss:
0.3307
Epoch 3/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.8832 - loss:
0.3149
Epoch 4/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.8842 - loss:
0.3096
Epoch 5/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8919 - loss:
0.2893
Epoch 6/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8974 - loss:
0.2800
```

```
Epoch 7/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8973 - loss:
0.2723
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8730 - loss:
0.3648

results

[0.3668724298477173, 0.8754799962043762]
```

Another attempt with 32 units and dropout

```
# building the model
model = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(32, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])

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

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=12,
                    batch_size=512,
                    validation_data=(x_val, y_val))

Epoch 1/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 73ms/step - accuracy: 0.5898 - loss:
0.6618 - val_accuracy: 0.8486 - val_loss: 0.4806
Epoch 2/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.7973 - loss:
0.4873 - val_accuracy: 0.8742 - val_loss: 0.3582
Epoch 3/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.8579 - loss:
0.3796 - val_accuracy: 0.8854 - val_loss: 0.3024
Epoch 4/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 0.8921 - loss:
0.3026 - val_accuracy: 0.8884 - val_loss: 0.2839
Epoch 5/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9172 - loss:
0.2486 - val_accuracy: 0.8906 - val_loss: 0.2737
```

```
Epoch 6/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.9288 - loss:
0.2189 - val_accuracy: 0.8860 - val_loss: 0.2799
Epoch 7/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9432 - loss:
0.1754 - val_accuracy: 0.8867 - val_loss: 0.2868
Epoch 8/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9496 - loss:
0.1555 - val_accuracy: 0.8862 - val_loss: 0.3009
Epoch 9/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 44ms/step - accuracy: 0.9581 - loss:
0.1357 - val_accuracy: 0.8848 - val_loss: 0.3125
Epoch 10/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9673 - loss:
0.1130 - val_accuracy: 0.8837 - val_loss: 0.3444
Epoch 11/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9722 - loss:
0.0954 - val_accuracy: 0.8860 - val_loss: 0.3602
Epoch 12/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 56ms/step - accuracy: 0.9709 - loss:
0.0902 - val_accuracy: 0.8838 - val_loss: 0.3944
```

```
model.fit(x_train, y_train, epochs=6, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9400 - loss:
0.1911
Epoch 2/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9471 - loss:
0.1634
Epoch 3/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 33ms/step - accuracy: 0.9556 - loss:
0.1423
Epoch 4/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.9615 - loss:
0.1209
Epoch 5/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9667 - loss:
0.1073
Epoch 6/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9702 - loss:
0.0941
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8703 - loss:
0.4777
```

```
results
```

```
[0.4730146527290344, 0.872759997844696]
```

An attempt with 1 layer, 16 units:

```python
# building the model
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])

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

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=12,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 89ms/step - accuracy: 0.6753 - loss: 
0.6062 - val_accuracy: 0.8630 - val_loss: 0.4278
Epoch 2/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.8385 - loss: 
0.4165 - val_accuracy: 0.8783 - val_loss: 0.3522
Epoch 3/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.8752 - loss: 
0.3408 - val_accuracy: 0.8832 - val_loss: 0.3148
Epoch 4/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8967 - loss: 
0.2866 - val_accuracy: 0.8883 - val_loss: 0.2931
Epoch 5/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9095 - loss: 
0.2548 - val_accuracy: 0.8846 - val_loss: 0.2905
Epoch 6/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9224 - loss: 
0.2330 - val_accuracy: 0.8881 - val_loss: 0.2790
Epoch 7/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9284 - loss: 
0.2155 - val_accuracy: 0.8897 - val_loss: 0.2723
Epoch 8/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.9419 - loss: 
0.1897 - val_accuracy: 0.8892 - val_loss: 0.2701
Epoch 9/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9413 - loss: 
0.1801 - val_accuracy: 0.8877 - val_loss: 0.2728
```

```
Epoch 10/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9467 - loss:
0.1654 - val_accuracy: 0.8887 - val_loss: 0.2746
Epoch 11/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9531 - loss:
0.1581 - val_accuracy: 0.8860 - val_loss: 0.2875
Epoch 12/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9569 - loss:
0.1405 - val_accuracy: 0.8869 - val_loss: 0.2800
```

```
model.fit(x_train, y_train, epochs=7, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9278 - loss:
0.2102
Epoch 2/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9392 - loss:
0.1829
Epoch 3/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9469 - loss:
0.1703
Epoch 4/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 30ms/step - accuracy: 0.9495 - loss:
0.1585
Epoch 5/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9556 - loss:
0.1443
Epoch 6/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9575 - loss:
0.1360
Epoch 7/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9607 - loss:
0.1269
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8774 - loss:
0.3306
```

```
results
```

```
[0.32845932245254517, 0.8791599869728088]
```

Attempt with 1 layer, 8 units

```
# building the model
model = keras.Sequential([
    layers.Dense(8, activation="relu"),
    layers.Dropout(0.5),
```

```
    layers.Dense(1, activation="sigmoid")
])

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

# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=12,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - accuracy: 0.6080 - loss:
0.6557 - val_accuracy: 0.8411 - val_loss: 0.5289
Epoch 2/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.7892 - loss:
0.5144 - val_accuracy: 0.8598 - val_loss: 0.4456
Epoch 3/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8201 - loss:
0.4432 - val_accuracy: 0.8744 - val_loss: 0.3907
Epoch 4/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8462 - loss:
0.3914 - val_accuracy: 0.8820 - val_loss: 0.3539
Epoch 5/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.8647 - loss:
0.3533 - val_accuracy: 0.8862 - val_loss: 0.3267
Epoch 6/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8701 - loss:
0.3288 - val_accuracy: 0.8843 - val_loss: 0.3157
Epoch 7/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8807 - loss:
0.3010 - val_accuracy: 0.8889 - val_loss: 0.2981
Epoch 8/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.8904 - loss:
0.2817 - val_accuracy: 0.8896 - val_loss: 0.2866
Epoch 9/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.8918 - loss:
0.2719 - val_accuracy: 0.8897 - val_loss: 0.2812
Epoch 10/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9017 - loss:
0.2507 - val_accuracy: 0.8890 - val_loss: 0.2748
Epoch 11/12
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9029 - loss:
0.2346 - val_accuracy: 0.8887 - val_loss: 0.2726
Epoch 12/12
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9066 - loss:
0.2326 - val_accuracy: 0.8888 - val_loss: 0.2708
```

```
model.fit(x_train, y_train, epochs=7, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8915 - loss:
0.2598
Epoch 2/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8988 - loss:
0.2433
Epoch 3/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9040 - loss:
0.2373
Epoch 4/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9092 - loss:
0.2183
Epoch 5/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.9081 - loss:
0.2172
Epoch 6/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9134 - loss:
0.2091
Epoch 7/7
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9161 - loss:
0.1999
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8798 - loss:
0.2920
```

```
results
```

```
[0.28911659121513367, 0.8827599883079529]
```

Attempt with 1 layer, 4 units

```
# building the model
model = keras.Sequential([
    layers.Dense(4, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])

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

```python
# Training the model
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=18,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 67ms/step - accuracy: 0.5849 - loss:
0.6567 - val_accuracy: 0.8221 - val_loss: 0.5391
Epoch 2/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.7348 - loss:
0.5370 - val_accuracy: 0.8617 - val_loss: 0.4689
Epoch 3/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.7803 - loss:
0.4843 - val_accuracy: 0.8766 - val_loss: 0.4305
Epoch 4/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8144 - loss:
0.4438 - val_accuracy: 0.8827 - val_loss: 0.3930
Epoch 5/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8259 - loss:
0.4207 - val_accuracy: 0.8824 - val_loss: 0.3708
Epoch 6/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.8474 - loss:
0.3939 - val_accuracy: 0.8837 - val_loss: 0.3573
Epoch 7/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.8489 - loss:
0.3852 - val_accuracy: 0.8871 - val_loss: 0.3431
Epoch 8/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.8660 - loss:
0.3636 - val_accuracy: 0.8877 - val_loss: 0.3367
Epoch 9/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.8659 - loss:
0.3511 - val_accuracy: 0.8822 - val_loss: 0.3292
Epoch 10/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.8726 - loss:
0.3430 - val_accuracy: 0.8881 - val_loss: 0.3145
Epoch 11/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.8750 - loss:
0.3324 - val_accuracy: 0.8875 - val_loss: 0.3143
Epoch 12/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.8756 - loss:
0.3246 - val_accuracy: 0.8869 - val_loss: 0.3082
Epoch 13/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8871 - loss:
0.3115 - val_accuracy: 0.8867 - val_loss: 0.3042
Epoch 14/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8956 - loss:
0.3033 - val_accuracy: 0.8886 - val_loss: 0.2933
Epoch 15/18
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8918 - loss:
0.2984 - val_accuracy: 0.8880 - val_loss: 0.2902
Epoch 16/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.8967 - loss:
0.2909 - val_accuracy: 0.8850 - val_loss: 0.3024
Epoch 17/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8911 - loss:
0.2906 - val_accuracy: 0.8871 - val_loss: 0.2912
Epoch 18/18
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.8995 - loss:
0.2790 - val_accuracy: 0.8864 - val_loss: 0.2885
```

```
model.fit(x_train, y_train, epochs=12, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8722 - loss:
0.3320
Epoch 2/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8772 - loss:
0.3173
Epoch 3/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8852 - loss:
0.3025
Epoch 4/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.8816 - loss:
0.3046
Epoch 5/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8909 - loss:
0.2885
Epoch 6/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.8903 - loss:
0.2846
Epoch 7/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.8938 - loss:
0.2779
Epoch 8/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.8988 - loss:
0.2672
Epoch 9/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.8969 - loss:
0.2680
Epoch 10/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8955 - loss:
0.2655
Epoch 11/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8929 - loss:
0.2645
Epoch 12/12
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9000 - loss:
```

```
0.2579
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8758 - loss:
0.3213

results

[0.32117849588394165, 0.8783599734306335]
```