Assignment-1 Neural Networks

##Installing the TensorFlow library using pip in a Jupyter Notebook

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.70.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.1
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages (from keras>=3.5.0->tensorflow) (0.14.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (202
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow) (3.
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from werkzeug>=1.0.1->tensorboard<2.19,>=2.
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow) (3
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->te
```

Load the IMDB dataset, selecting the 10,000 most commonly occurring words, for use in training and testing a sentiment analysis model.

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

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

# Displaying the train_data along with its shape (dimensions)

```
print(train_data,train_data.shape)
```

```
[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, 4
 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,
 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,
 ...
 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,
 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, 1
 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
```

```
train_labels[0] ## Accessing the first label in the train_labels dataset
len(train_labels) ## Getting the total number of labels in the training dataset
test_labels[0] ## Accessing the first label in the test_labels dataset
max([max(sequence_647) for sequence_647 in test_data]) ## Finding the maximum word index in the test_data sequences
```

```
9999
```

Convert the word indices back to words using the reverse word index to decode the first review from the training dataset.

```
word_index_647 = imdb.get_word_index()
reverse_word_index_647 = dict(
    [(value, key) for (key, value) in word_index_647.items()])
decoded_review = " ".join(
    [reverse_word_index_647.get(i - 3, "?") for i in train_data[0]])
```

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

Transforms each sequence of integers into a binary vector of a defined dimension, setting the corresponding index to 1 if the word appears in the sequence.

```
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) # Converting training data into binary vector representations
x_test = vectorize_sequences(test_data) # Converting test data into binary vector representations
```

```
x_train[0] # Displaying the first vectorized review from the training dataset
```

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

```
x_test[0] # Displaying the first vectorized review from the test data
```

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

```
y_train = np.asarray(train_labels).astype("float32") # Converting training labels into a NumPy array with float32 data type
y_test = np.asarray(test_labels).astype("float32") # Converting test labels into a NumPy array with float32 data type
```

# Creating a neural network model with three layers: two hidden layers with ReLU activation and an output layer with a sigmoid activation for binary classification.

```
from tensorflow import keras
from tensorflow.keras import layers

model647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
```

# Compiling the model with RMSprop optimizer, binary cross-entropy loss function, and accuracy as the evaluation metric.

```
model647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

Splitting the training data into validation and training sets, using the first 10,000 samples for validation.

```
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 for 20 epochs with a batch size of 512 and using validation data for monitoring performance.

```
## model planned to train with 20 epoch with batch size of 256
history = model647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 68ms/step - accuracy: 0.7184 - loss: 0.5933 - val_accuracy: 0.8653 - val_loss: 0.3948
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.8916 - loss: 0.3392 - val_accuracy: 0.8791 - val_loss: 0.3136
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - accuracy: 0.9177 - loss: 0.2497 - val_accuracy: 0.8797 - val_loss: 0.2978
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 46ms/step - accuracy: 0.9350 - loss: 0.1985 - val_accuracy: 0.8895 - val_loss: 0.2750
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.9508 - loss: 0.1578 - val_accuracy: 0.8824 - val_loss: 0.2922
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9548 - loss: 0.1415 - val_accuracy: 0.8854 - val_loss: 0.2917
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9636 - loss: 0.1179 - val_accuracy: 0.8864 - val_loss: 0.2943
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9697 - loss: 0.1007 - val_accuracy: 0.8833 - val_loss: 0.3080
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9746 - loss: 0.0900 - val_accuracy: 0.8791 - val_loss: 0.3319
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9791 - loss: 0.0781 - val_accuracy: 0.8741 - val_loss: 0.3584
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9844 - loss: 0.0638 - val_accuracy: 0.8765 - val_loss: 0.3679
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9870 - loss: 0.0555 - val_accuracy: 0.8801 - val_loss: 0.3859
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9901 - loss: 0.0486 - val_accuracy: 0.8751 - val_loss: 0.4053
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9912 - loss: 0.0406 - val_accuracy: 0.8755 - val_loss: 0.4272
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9938 - loss: 0.0325 - val_accuracy: 0.8761 - val_loss: 0.4468
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9953 - loss: 0.0279 - val_accuracy: 0.8745 - val_loss: 0.4653
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9974 - loss: 0.0225 - val_accuracy: 0.8730 - val_loss: 0.4897
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9988 - loss: 0.0182 - val_accuracy: 0.8594 - val_loss: 0.6004
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9974 - loss: 0.0201 - val_accuracy: 0.8583 - val_loss: 0.5925
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9973 - loss: 0.0187 - val_accuracy: 0.8689 - val_loss: 0.5607
```

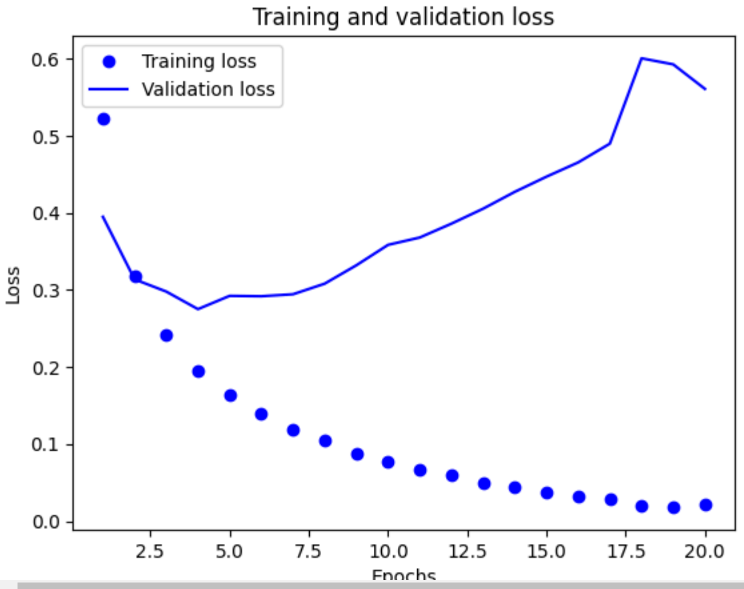
# Retrieving and displaying the keys of the history dictionary to access training and validation metrics.

```
history_dict647 = history.history
history_dict647.keys()
```

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

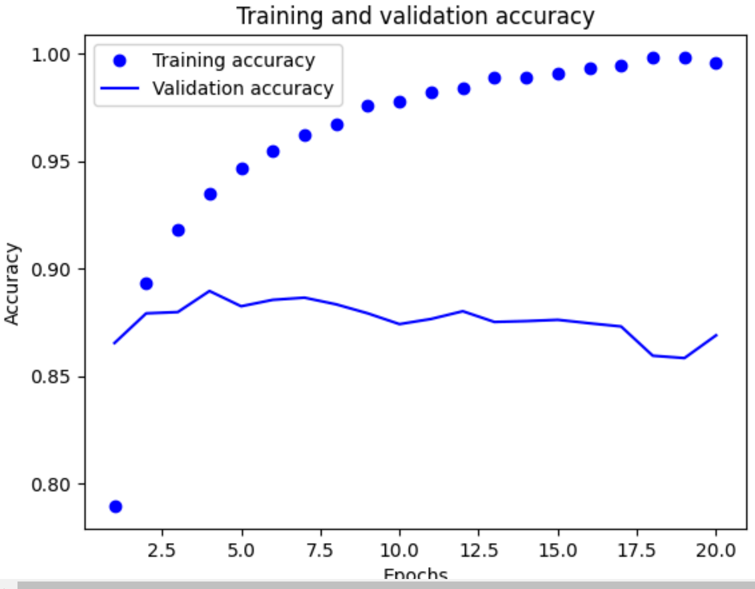
# Plotting the training and validation loss over epochs to visualize model performance during training.

```
#Plotting the training loss vs validation loss
import matplotlib.pyplot as plot647
history_dict647 = history.history
loss_values = history_dict647["loss"]
val_loss_values = history_dict647["val_loss"]
epochs = range(1, len(loss_values) + 1)
plot647.plot(epochs, loss_values, "bo", label="Training loss")
plot647.plot(epochs, val_loss_values, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()
```



# Plotting the training and validation accuracy over epochs to evaluate the model's performance on both datasets.

```
#Plotting training accuracy vs validatition accuracy
plot647.clf()
acc = history_dict647["accuracy"]
val_acc = history_dict647["val_accuracy"]
plot647.plot(epochs, acc, "bo", label="Training accuracy")
plot647.plot(epochs, val_acc, "b", label="Validation accuracy")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```



# Defining, compiling, and training a neural network model with two hidden layers, then evaluating its performance on the test data.

```
model647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

```
model647.fit(x_train, y_train, epochs=4, batch_size=512)
results = model647.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.7321 - loss: 0.5830
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 31ms/step - accuracy: 0.9000 - loss: 0.3073
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.9183 - loss: 0.2293
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9334 - loss: 0.1907
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8862 - loss: 0.2828
```

```
results #Displaying the results
```

```
[0.28102514147758484, 0.888480007648468]
```

# Using the trained model to make predictions on the test data.

```
model647.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.23334053],
       [0.9991537 ],
       [0.87785053],
       ...,
       [0.13598225],
       [0.09242532],
       [0.6118001 ]], dtype=float32)
```

# Defining, compiling, and training a simplified neural network model with one hidden layer, and using a validation set for performance monitoring.

```
model_647_layer = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model_647_layer.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

x_val647 = x_train[:10000]
partial_x_train = x_train[10000:]

y_val647 = y_train[:10000]
partial_y_train = y_train[10000:]


history_layer647 = model_647_layer.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val647, y_val647))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 0.7120 - loss: 0.5766 - val_accuracy: 0.8737 - val_loss: 0.3871
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.9010 - loss: 0.3321 - val_accuracy: 0.8841 - val_loss: 0.3180
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.9186 - loss: 0.2602 - val_accuracy: 0.8851 - val_loss: 0.2967
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9352 - loss: 0.2152 - val_accuracy: 0.8894 - val_loss: 0.2794
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9423 - loss: 0.1888 - val_accuracy: 0.8880 - val_loss: 0.2786
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9492 - loss: 0.1681 - val_accuracy: 0.8862 - val_loss: 0.2810
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9557 - loss: 0.1478 - val_accuracy: 0.8860 - val_loss: 0.2771
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9617 - loss: 0.1328 - val_accuracy: 0.8865 - val_loss: 0.2852
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9666 - loss: 0.1186 - val_accuracy: 0.8849 - val_loss: 0.2926
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9704 - loss: 0.1113 - val_accuracy: 0.8844 - val_loss: 0.2946
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9744 - loss: 0.1001 - val_accuracy: 0.8841 - val_loss: 0.3034
```

```
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9769 - loss: 0.0927 - val_accuracy: 0.8797 - val_loss: 0.3134
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9802 - loss: 0.0848 - val_accuracy: 0.8767 - val_loss: 0.3269
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9845 - loss: 0.0757 - val_accuracy: 0.8814 - val_loss: 0.3259
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9876 - loss: 0.0702 - val_accuracy: 0.8799 - val_loss: 0.3340
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.9887 - loss: 0.0629 - val_accuracy: 0.8787 - val_loss: 0.3433
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9885 - loss: 0.0590 - val_accuracy: 0.8751 - val_loss: 0.3566
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9911 - loss: 0.0553 - val_accuracy: 0.8770 - val_loss: 0.3621
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9917 - loss: 0.0496 - val_accuracy: 0.8772 - val_loss: 0.3717
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9944 - loss: 0.0446 - val_accuracy: 0.8770 - val_loss: 0.3905
```

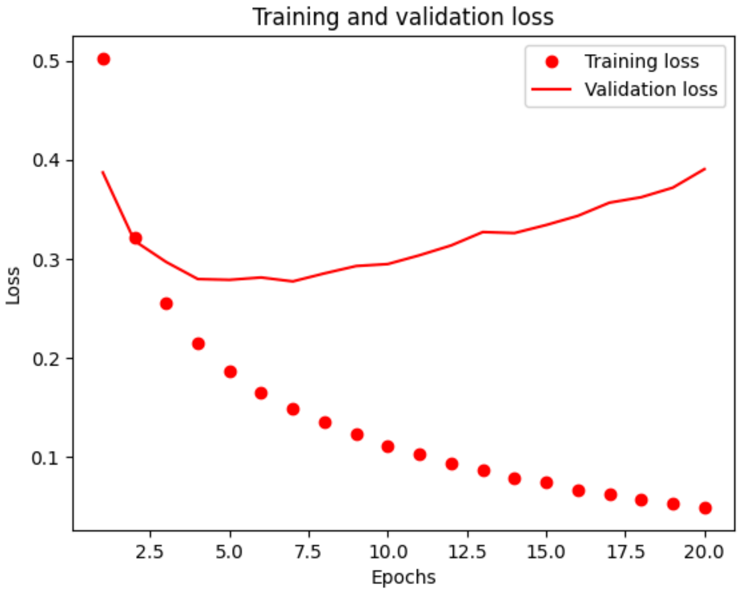
Retrieve the training history from history_layer647 and show the keys of the history dictionary.

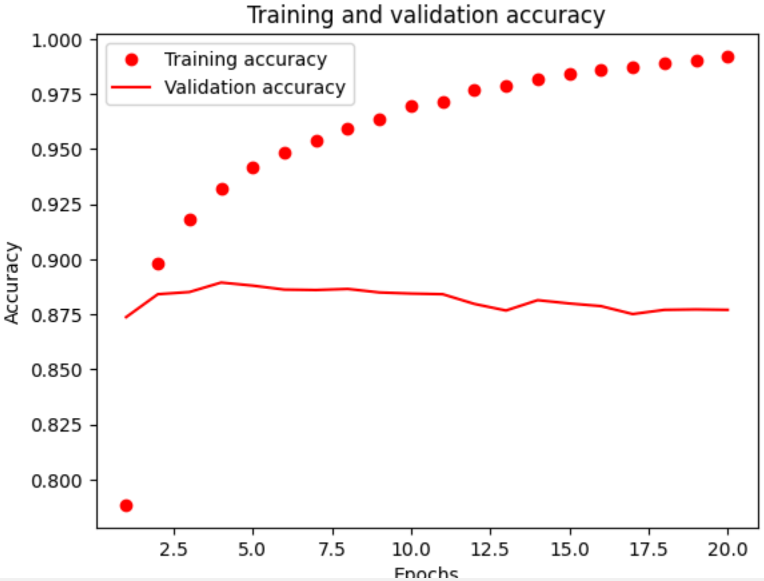
```python
history_dict647 = history_layer647.history
history_dict647.keys()
```

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

# Plotting the training and validation loss, followed by the training and validation accuracy over epochs to visualize model performance.

```python
import matplotlib.pyplot as plot647
history_dict647 = history_layer647.history
loss_value647 = history_dict647["loss"]
val_loss_value647 = history_dict647["val_loss"]
epochs647 = range(1, len(loss_value647) + 1)

#Plotting graph of Training and Validation loss
plot647.plot(epochs647, loss_value647, "ro", label="Training loss")
plot647.plot(epochs647, val_loss_value647, "r", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()

#Plotting graph of Training and Validation Accuracy
plot647.clf()
accuracy647 = history_dict647["accuracy"]
val_accuracy1 = history_dict647["val_accuracy"]
plot647.plot(epochs647, accuracy647, "ro", label="Training accuracy")
plot647.plot(epochs647, val_accuracy1, "r", label="Validation accuracy")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





# Defining, compiling, and training a simplified neural network model for binary classification, followed by evaluating its performance on the test data.

```
model_647_layer = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

model_647_layer.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model_647_layer.fit(x_train, y_train, epochs=5, batch_size=512)
result_647_layer = model_647_layer.evaluate(x_test, y_test)
```

```
Epoch 1/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.7509 - loss: 0.5489
Epoch 2/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.8972 - loss: 0.3098
Epoch 3/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9200 - loss: 0.2383
Epoch 4/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9291 - loss: 0.2095
Epoch 5/5
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9386 - loss: 0.1825
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8853 - loss: 0.2802
```

# Printing the evaluation results (loss and accuracy) of the model on the test data.

```
print(result_647_layer)
```

```
[0.27949032187461853, 0.8871999979019165]
```

# Using the trained model to make predictions on the test data.

```
model_647_layer.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
array([[0.20644897],
       [0.9996556 ],
       [0.59752214],
       ...,
       [0.10031941],
       [0.07974253],
       [0.47744647]], dtype=float32)
```

# Defining, compiling, and training a neural network model with three hidden layers, and using a validation set for performance monitoring.

```
model_3_layers_647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_3_layers_647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
x_val3_647 = x_train[:10000]
partial_x_train_647 = x_train[10000:]

y_val3_647 = y_train[:10000]
partial_y_train_647 = y_train[10000:]

history_3_layers_647 = model_3_layers_647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val3_647, y_val3_647))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 67ms/step - accuracy: 0.6264 - loss: 0.6322 - val_accuracy: 0.8568 - val_loss: 0.3973
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.8913 - loss: 0.3313 - val_accuracy: 0.8861 - val_loss: 0.3035
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9215 - loss: 0.2347 - val_accuracy: 0.8862 - val_loss: 0.2841
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9415 - loss: 0.1799 - val_accuracy: 0.8778 - val_loss: 0.3033
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9558 - loss: 0.1396 - val_accuracy: 0.8547 - val_loss: 0.3942
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9639 - loss: 0.1165 - val_accuracy: 0.8765 - val_loss: 0.3220
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9719 - loss: 0.0938 - val_accuracy: 0.8783 - val_loss: 0.3481
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9806 - loss: 0.0781 - val_accuracy: 0.8720 - val_loss: 0.3961
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9830 - loss: 0.0686 - val_accuracy: 0.8816 - val_loss: 0.3676
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9898 - loss: 0.0469 - val_accuracy: 0.8798 - val_loss: 0.3889
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9929 - loss: 0.0383 - val_accuracy: 0.8773 - val_loss: 0.4242
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9952 - loss: 0.0287 - val_accuracy: 0.8783 - val_loss: 0.4472
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9978 - loss: 0.0208 - val_accuracy: 0.8699 - val_loss: 0.5397
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9807 - loss: 0.0555 - val_accuracy: 0.8739 - val_loss: 0.4910
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9947 - loss: 0.0225 - val_accuracy: 0.8751 - val_loss: 0.5179
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9983 - loss: 0.0123 - val_accuracy: 0.8729 - val_loss: 0.5414
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9996 - loss: 0.0074 - val_accuracy: 0.8644 - val_loss: 0.5986
Epoch 18/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9980 - loss: 0.0113 - val_accuracy: 0.8739 - val_loss: 0.6084
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 0.8767 - val_loss: 0.6434
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9960 - loss: 0.0139 - val_accuracy: 0.8727 - val_loss: 0.6592
```

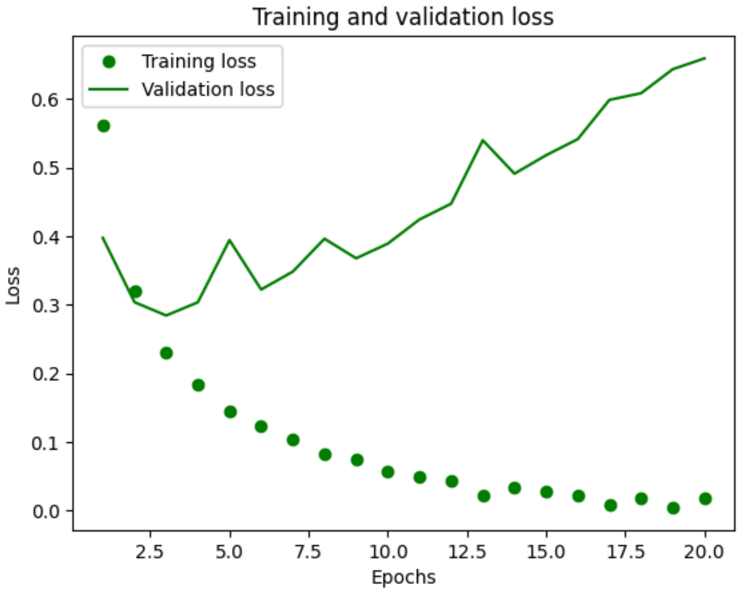
# Extracting and displaying the keys of the training history dictionary for the model with three hidden layers.

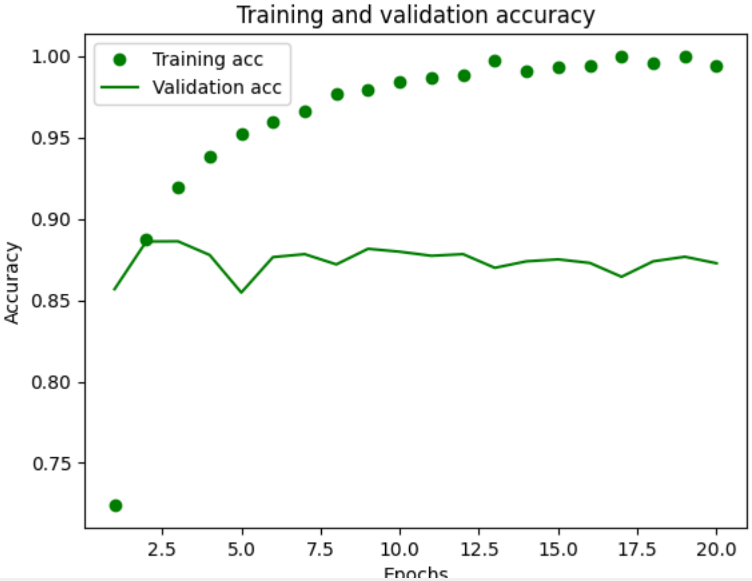
```
history_dict_3_647 = history_3_layers_647.history
history_dict_3_647.keys()
```

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

# Plotting the training and validation loss, followed by the training and validation accuracy over epochs for the model with three hidden layers.

```
loss_val647 = history_dict_3_647["loss"]
val_loss_val3 = history_dict_3_647["val_loss"]
epochs3 = range(1, len(loss_val647) + 1)
plot647.plot(epochs3, loss_val647, "go", label="Training loss")
plot647.plot(epochs3, val_loss_val3, "g", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()


plot647.clf()
accuracy3 = history_dict_3_647["accuracy"]
val_accuracy3 = history_dict_3_647["val_accuracy"]
plot647.plot(epochs3, accuracy3, "go", label="Training acc")
plot647.plot(epochs3, val_accuracy3, "g", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





# Defining, compiling, and training a neural network model with three hidden layers, followed by evaluating its performance on the test data.

```
model_3_layers_647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])


model_3_layers_647.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model_3_layers_647.fit(x_train, y_train, epochs=3, batch_size=512)
results_3_layers = model_3_layers_647.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 36ms/step - accuracy: 0.7183 - loss: 0.5715
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9033 - loss: 0.2742
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9293 - loss: 0.2030
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8860 - loss: 0.2860
```

```
print(result_647_layer) # Printing the test evaluation results (loss and accuracy) of the previously trained model.
```

```
[0.27949032187461853, 0.8871999979019165]
```

```
model_647_layer.predict(x_test) # Generating predictions on the test data using the trained model.
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.20644897],
       [0.9996556 ],
       [0.59752214],
       ...,
       [0.10031941],
       [0.07974253],
       [0.47744647]], dtype=float32)
```

# Building, compiling, and training a neural network with three hidden layers, using a validation set to monitor performance over 20 epochs.

```
model_3_layers_647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_3_layers_647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
x_val3_647 = x_train[:10000]
partial_x_train_647 = x_train[10000:]

y_val3_647 = y_train[:10000]
partial_y_train_647 = y_train[10000:]

history_3_layers_647 = model_3_layers_647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val3_647, y_val3_647))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 66ms/step - accuracy: 0.6555 - loss: 0.6378 - val_accuracy: 0.8562 - val_loss: 0.4364
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.8821 - loss: 0.3805 - val_accuracy: 0.8594 - val_loss: 0.3523
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9129 - loss: 0.2678 - val_accuracy: 0.8891 - val_loss: 0.2867
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9348 - loss: 0.1993 - val_accuracy: 0.8873 - val_loss: 0.2831
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9478 - loss: 0.1632 - val_accuracy: 0.8827 - val_loss: 0.2904
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9557 - loss: 0.1357 - val_accuracy: 0.8821 - val_loss: 0.3008
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 64ms/step - accuracy: 0.9669 - loss: 0.1087 - val_accuracy: 0.8868 - val_loss: 0.3010
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9756 - loss: 0.0906 - val_accuracy: 0.8846 - val_loss: 0.3243
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9821 - loss: 0.0670 - val_accuracy: 0.8826 - val_loss: 0.3433
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9873 - loss: 0.0545 - val_accuracy: 0.8628 - val_loss: 0.4190
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9889 - loss: 0.0444 - val_accuracy: 0.8771 - val_loss: 0.3890
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9938 - loss: 0.0324 - val_accuracy: 0.8766 - val_loss: 0.4103
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9969 - loss: 0.0239 - val_accuracy: 0.8768 - val_loss: 0.4411
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9967 - loss: 0.0196 - val_accuracy: 0.8751 - val_loss: 0.4610
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9984 - loss: 0.0138 - val_accuracy: 0.8739 - val_loss: 0.4833
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9993 - loss: 0.0092 - val_accuracy: 0.8737 - val_loss: 0.5144
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9971 - loss: 0.0131 - val_accuracy: 0.8734 - val_loss: 0.5405
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9986 - loss: 0.0081 - val_accuracy: 0.8718 - val_loss: 0.5508
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9999 - loss: 0.0043 - val_accuracy: 0.8720 - val_loss: 0.5791
Epoch 20/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9992 - loss: 0.0049 - val_accuracy: 0.8710 - val_loss: 0.5948
```

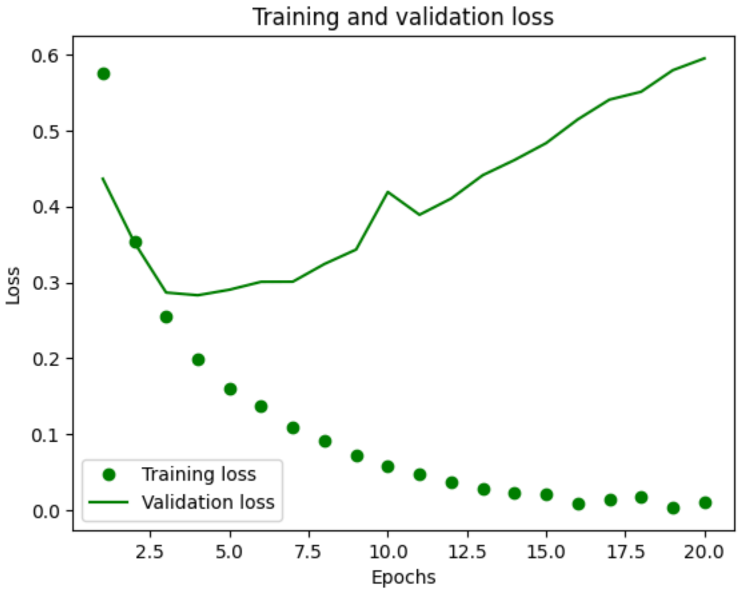
# Extracting and displaying the keys of the training history dictionary for the three-layer model.

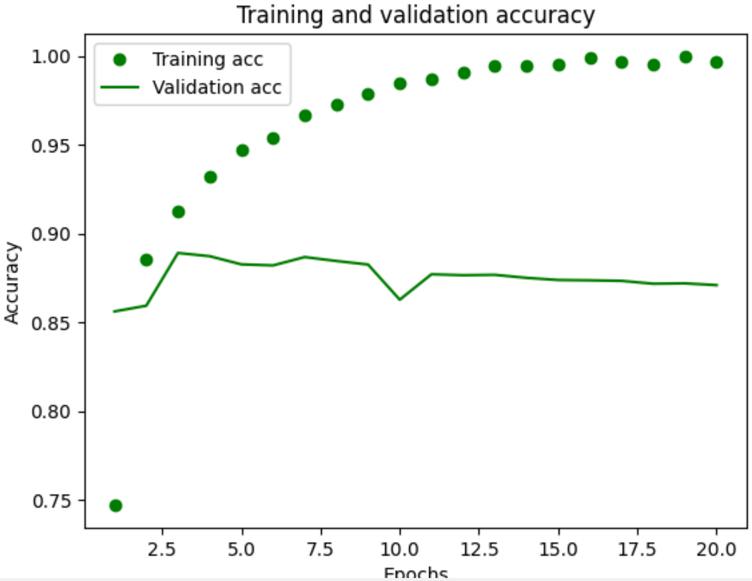
```
history_dict_3_647 = history_3_layers_647.history
history_dict_3_647.keys()
```

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

# Plotting training and validation loss, followed by training and validation accuracy over epochs for the three-layer model.

```
loss_val647 = history_dict_3_647["loss"]
val_loss_val3 = history_dict_3_647["val_loss"]
epochs3 = range(1, len(loss_val647) + 1)
plot647.plot(epochs3, loss_val647, "go", label="Training loss")
plot647.plot(epochs3, val_loss_val3, "g", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()


plot647.clf()
accuracy3 = history_dict_3_647["accuracy"]
val_accuracy3 = history_dict_3_647["val_accuracy"]
plot647.plot(epochs3, accuracy3, "go", label="Training acc")
plot647.plot(epochs3, val_accuracy3, "g", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





# Defining, compiling, training a three-hidden-layer neural network for three epochs, and evaluating its performance on the test data.

```
model_3_layers_647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])


model_3_layers_647.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model_3_layers_647.fit(x_train, y_train, epochs=3, batch_size=512)
results_3_layers = model_3_layers_647.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.7151 - loss: 0.5616
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9080 - loss: 0.2649
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.9334 - loss: 0.1924
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8724 - loss: 0.3203
```

```
print(results_3_layers) # Printing the evaluation results (loss and accuracy) of the three-layer model on the test data.
```

```
[0.3158565163612366, 0.8740400075912476]
```

```
model_3_layers_647.predict(x_test) # Generating predictions on the test data using the trained three-layer model.
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.2887889 ],
       [0.9992645 ],
       [0.99066985],
       ...,
       [0.18958339],
       [0.14366959],
       [0.8326748 ]], dtype=float32)
```

# Building, compiling, and training a neural network with three hidden layers of 32 units each, using a validation set over 20 epochs.

```
model_32_units_647 = keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
#model compilation
model_32_units_647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
#model validation
x_val_32_647 = x_train[:10000]
partial_x_train = x_train[10000:]

y_val_32_647 = y_train[:10000]
partial_y_train = y_train[10000:]


history_32_units_647 = model_32_units_647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val_32_647, y_val_32_647))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 94ms/step - accuracy: 0.6641 - loss: 0.6107 - val_accuracy: 0.7929 - val_loss: 0.4460
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 46ms/step - accuracy: 0.8800 - loss: 0.3262 - val_accuracy: 0.8766 - val_loss: 0.3104
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - accuracy: 0.9150 - loss: 0.2341 - val_accuracy: 0.8906 - val_loss: 0.2746
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - accuracy: 0.9356 - loss: 0.1815 - val_accuracy: 0.8657 - val_loss: 0.3407
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.9487 - loss: 0.1470 - val_accuracy: 0.8858 - val_loss: 0.2942
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9566 - loss: 0.1218 - val_accuracy: 0.8811 - val_loss: 0.3147
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 57ms/step - accuracy: 0.9704 - loss: 0.0952 - val_accuracy: 0.8800 - val_loss: 0.3504
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9745 - loss: 0.0783 - val_accuracy: 0.8788 - val_loss: 0.3649
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 0.9708 - loss: 0.0896 - val_accuracy: 0.8642 - val_loss: 0.4709
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9807 - loss: 0.0639 - val_accuracy: 0.8752 - val_loss: 0.4225
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - accuracy: 0.9796 - loss: 0.0646 - val_accuracy: 0.8754 - val_loss: 0.4423
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9877 - loss: 0.0409 - val_accuracy: 0.8738 - val_loss: 0.4984
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9929 - loss: 0.0304 - val_accuracy: 0.8757 - val_loss: 0.4939
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 56ms/step - accuracy: 0.9967 - loss: 0.0174 - val_accuracy: 0.8738 - val_loss: 0.5115
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9992 - loss: 0.0113 - val_accuracy: 0.8732 - val_loss: 0.5808
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 68ms/step - accuracy: 0.9915 - loss: 0.0334 - val_accuracy: 0.8718 - val_loss: 0.5817
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9984 - loss: 0.0099 - val_accuracy: 0.8727 - val_loss: 0.6004
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9994 - loss: 0.0060 - val_accuracy: 0.8714 - val_loss: 0.6498
Epoch 19/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - accuracy: 0.9981 - loss: 0.0098 - val_accuracy: 0.8700 - val_loss: 0.6655
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9999 - loss: 0.0031 - val_accuracy: 0.8706 - val_loss: 0.6998
```

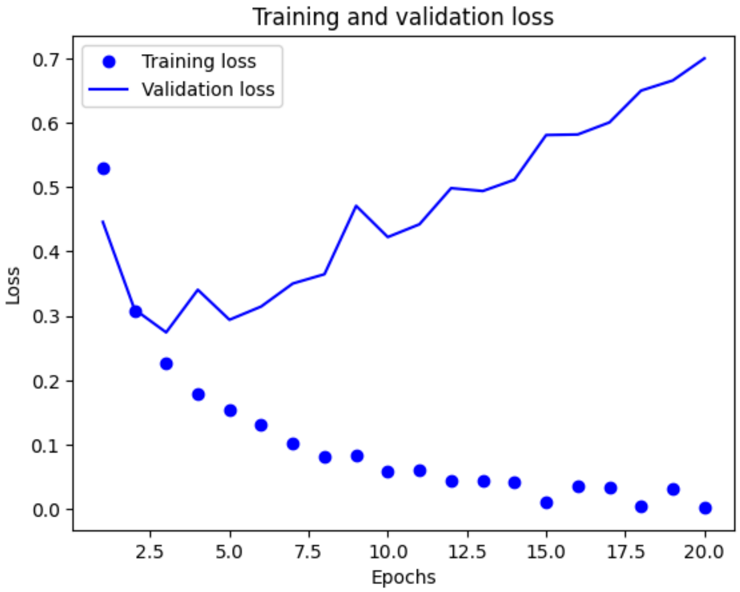
# Extracting the training history from the model and displaying the available metrics recorded during training.

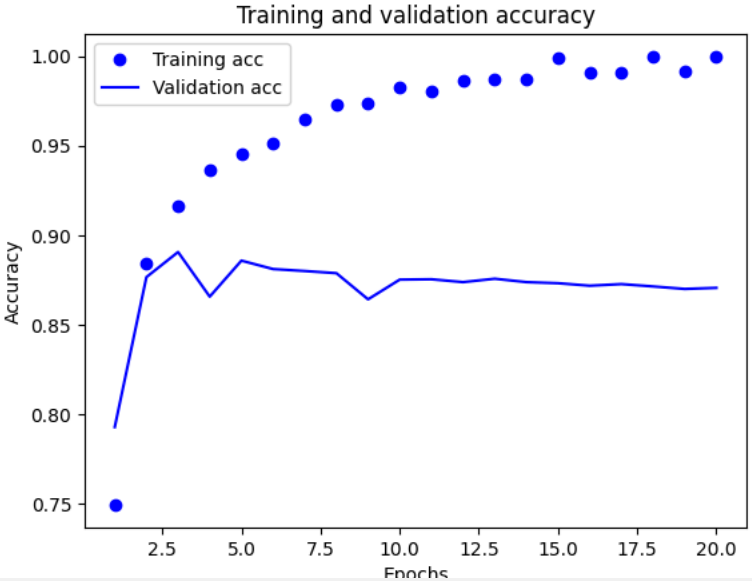
```
history_dict_32_647 = history_32_units_647.history
history_dict_32_647.keys()
```

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

# Plotting the training and validation loss, followed by the training and validation accuracy over epochs for the model with 32 units in each layer.

```
loss_value_32_647 = history_dict_32_647["loss"]
val_loss_value_32_647 = history_dict_32_647["val_loss"]
epochs_32 = range(1, len(loss_value_32_647) + 1)
plot647.plot(epochs_32, loss_value_32_647, "bo", label="Training loss")
plot647.plot(epochs_32, val_loss_value_32_647, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()


plot647.clf()
accuracy_32 = history_dict_32_647["accuracy"]
val_accuracy_32 = history_dict_32_647["val_accuracy"]
plot647.plot(epochs_32, accuracy_32, "bo", label="Training acc")
plot647.plot(epochs_32, val_accuracy_32, "b", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





```
history_32_units_647 = model_32_units_647.fit(x_train, y_train, epochs=3, batch_size=512) # Training the model with 32 units in each layer f
results_32_units_647 = model_32_units_647.evaluate(x_test, y_test) # Evaluating the performance of the model on the test data and storing th
results_32_units_647 # Displaying the evaluation results (loss and accuracy) of the model on the test data.
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.9377 - loss: 0.2603
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.9628 - loss: 0.1225
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9742 - loss: 0.0813
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8646 - loss: 0.4105
[0.40794897079467773, 0.8654400110244751]
```

# Building, compiling, and training a neural network with two hidden layers of 64 units each, using a validation set over 20 epochs.

```
model_64_units_647 = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_64_units_647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_val_64_2 = x_train[:10000]
```

```
partial_x_train_64_2 = x_train[10000:]

y_val_64_2 = y_train[:10000]
partial_y_train_64_2 = y_train[10000:]

history_64_647 = model_64_units_647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val_64_2, y_val_64_2))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 126ms/step - accuracy: 0.7081 - loss: 0.5794 - val_accuracy: 0.8703 - val_loss: 0.3420
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 65ms/step - accuracy: 0.8771 - loss: 0.3146 - val_accuracy: 0.8795 - val_loss: 0.2939
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.9217 - loss: 0.2123 - val_accuracy: 0.8542 - val_loss: 0.3624
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9300 - loss: 0.1879 - val_accuracy: 0.8858 - val_loss: 0.2851
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.9506 - loss: 0.1415 - val_accuracy: 0.8784 - val_loss: 0.3201
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 94ms/step - accuracy: 0.9580 - loss: 0.1154 - val_accuracy: 0.8858 - val_loss: 0.3087
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.9710 - loss: 0.0898 - val_accuracy: 0.8822 - val_loss: 0.3338
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 69ms/step - accuracy: 0.9737 - loss: 0.0846 - val_accuracy: 0.8805 - val_loss: 0.3496
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9854 - loss: 0.0560 - val_accuracy: 0.8782 - val_loss: 0.3976
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 101ms/step - accuracy: 0.9866 - loss: 0.0489 - val_accuracy: 0.8700 - val_loss: 0.4252
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.9902 - loss: 0.0388 - val_accuracy: 0.8769 - val_loss: 0.4198
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.9954 - loss: 0.0238 - val_accuracy: 0.8796 - val_loss: 0.4307
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.9994 - loss: 0.0115 - val_accuracy: 0.8049 - val_loss: 0.9741
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 67ms/step - accuracy: 0.9693 - loss: 0.0954 - val_accuracy: 0.8778 - val_loss: 0.4868
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 98ms/step - accuracy: 0.9954 - loss: 0.0183 - val_accuracy: 0.8773 - val_loss: 0.4872
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 74ms/step - accuracy: 0.9999 - loss: 0.0050 - val_accuracy: 0.8765 - val_loss: 0.5345
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 74ms/step - accuracy: 0.9973 - loss: 0.0122 - val_accuracy: 0.8759 - val_loss: 0.5385
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9999 - loss: 0.0034 - val_accuracy: 0.8766 - val_loss: 0.5656
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 123ms/step - accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 0.8753 - val_loss: 0.6046
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9914 - loss: 0.0312 - val_accuracy: 0.8760 - val_loss: 0.5833
```

# Extracting the training history from the model with 64 units and displaying the available metrics recorded during training.

```
history_dict_64_647 = history_64_647.history
history_dict_64_647.keys()
```

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

# Building, compiling, and training a neural network with two hidden layers of 64 units each, using a validation set for 20 epochs.

```
model_64_units_647 = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_64_units_647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_val_64_2 = x_train[:10000]
partial_x_train_64_2 = x_train[10000:]

y_val_64_2 = y_train[:10000]
partial_y_train_64_2 = y_train[10000:]

history_64_647 = model_64_units_647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val_64_2, y_val_64_2))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 95ms/step - accuracy: 0.6845 - loss: 0.6016 - val_accuracy: 0.8523 - val_loss: 0.3760
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 103ms/step - accuracy: 0.8720 - loss: 0.3301 - val_accuracy: 0.8863 - val_loss: 0.2891
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 72ms/step - accuracy: 0.9146 - loss: 0.2283 - val_accuracy: 0.8783 - val_loss: 0.3011
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9284 - loss: 0.1909 - val_accuracy: 0.8834 - val_loss: 0.2858
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 65ms/step - accuracy: 0.9375 - loss: 0.1681 - val_accuracy: 0.8817 - val_loss: 0.3145
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 106ms/step - accuracy: 0.9550 - loss: 0.1271 - val_accuracy: 0.8853 - val_loss: 0.2987
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 69ms/step - accuracy: 0.9710 - loss: 0.0955 - val_accuracy: 0.8788 - val_loss: 0.3274
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 70ms/step - accuracy: 0.9726 - loss: 0.0826 - val_accuracy: 0.8755 - val_loss: 0.3510
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.9867 - loss: 0.0556 - val_accuracy: 0.8795 - val_loss: 0.3583
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 97ms/step - accuracy: 0.9906 - loss: 0.0432 - val_accuracy: 0.8778 - val_loss: 0.3822
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - accuracy: 0.9898 - loss: 0.0387 - val_accuracy: 0.8763 - val_loss: 0.4018
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 71ms/step - accuracy: 0.9963 - loss: 0.0242 - val_accuracy: 0.8772 - val_loss: 0.4243
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9990 - loss: 0.0138 - val_accuracy: 0.8705 - val_loss: 0.5418
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - accuracy: 0.9810 - loss: 0.0516 - val_accuracy: 0.8767 - val_loss: 0.4901
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 71ms/step - accuracy: 0.9948 - loss: 0.0218 - val_accuracy: 0.8762 - val_loss: 0.4949
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 68ms/step - accuracy: 1.0000 - loss: 0.0058 - val_accuracy: 0.8765 - val_loss: 0.5429
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 72ms/step - accuracy: 0.9945 - loss: 0.0203 - val_accuracy: 0.8750 - val_loss: 0.5306
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 91ms/step - accuracy: 0.9999 - loss: 0.0038 - val_accuracy: 0.8766 - val_loss: 0.5759
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 69ms/step - accuracy: 0.9967 - loss: 0.0121 - val_accuracy: 0.8724 - val_loss: 0.5717
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 0.8737 - val_loss: 0.5985
```

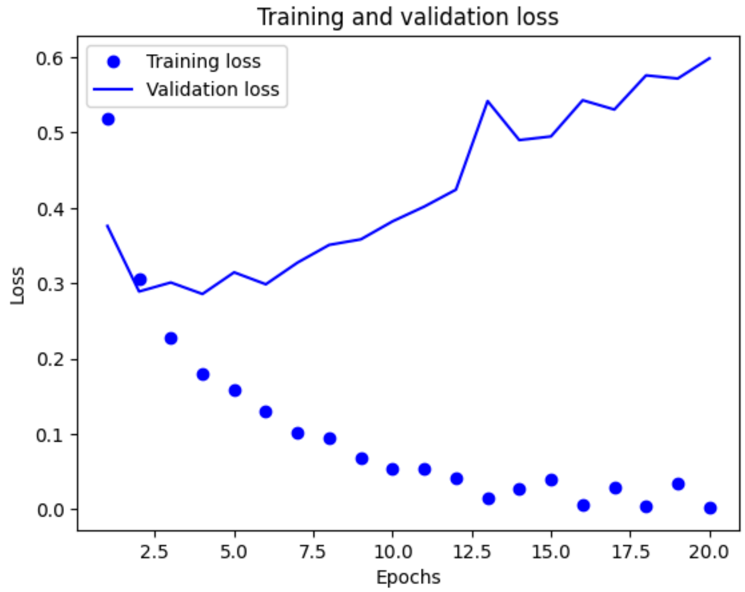
# Extracting and displaying the keys of the training history dictionary for the model with 64 units.

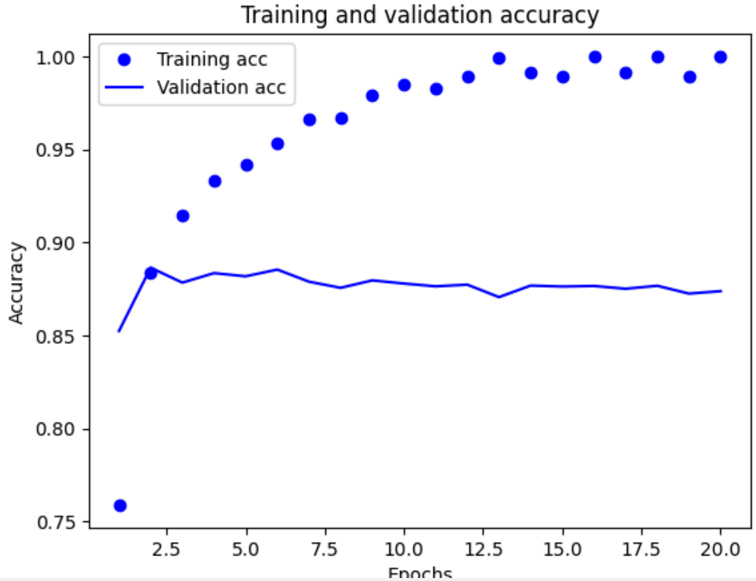
```
history_dict_64_647 = history_64_647.history
history_dict_64_647.keys()
```

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

# Plotting the training and validation loss, followed by the training and validation accuracy for the model with 64 units.

```
loss_value64 = history_dict_64_647["loss"]
val_loss_value64 = history_dict_64_647["val_loss"]
epochs_64 = range(1, len(loss_value64) + 1)
plot647.plot(epochs_64, loss_value64, "bo", label="Training loss")
plot647.plot(epochs_64, val_loss_value64, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()

plot647.clf()
accuracy_64 = history_dict_64_647["accuracy"]
val_accuracy_64 = history_dict_64_647["val_accuracy"]
plot647.plot(epochs_64, accuracy_64, "bo", label="Training acc")
plot647.plot(epochs_64, val_accuracy_64, "b", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





# Training the model with 64 units for 3 epochs and evaluating the performance on the test set.

```
history_64_647 = model_64_units_647.fit(x_train, y_train, epochs=3, batch_size=512)
results_64_units_647 = model_64_units_647.evaluate(x_test, y_test)
results_64_units_647
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - accuracy: 0.9468 - loss: 0.2084
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - accuracy: 0.9694 - loss: 0.0968
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9840 - loss: 0.0572
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8618 - loss: 0.4203
[0.4104676842689514, 0.8658000230789185]
```

```
model_64_units_647.predict(x_test) # Predicting the output using the trained model with 64 units on the test set.
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step
array([[0.056929  ],
       [0.99999994],
       [0.96342087],
       ...,
       [0.16129674],
       [0.0199403 ],
       [0.9147184 ]], dtype=float32)
```

# Creating and training a model with 128 units in each layer, validating with a subset of the training data.

```
model_128units_647 = keras.Sequential([
    layers.Dense(128, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model_128units_647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
# validation
x_val_128_647 = x_train[:10000]
partial_x_train_647 = x_train[10000:]

y_val_128_647 = y_train[:10000]
partial_y_train_647 = y_train[10000:]

history_128_3 = model_128units_647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val_128_647, y_val_128_647))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 167ms/step - accuracy: 0.6681 - loss: 0.6182 - val_accuracy: 0.7901 - val_loss: 0.4603
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 121ms/step - accuracy: 0.8705 - loss: 0.3283 - val_accuracy: 0.8056 - val_loss: 0.4444
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 119ms/step - accuracy: 0.9115 - loss: 0.2335 - val_accuracy: 0.8834 - val_loss: 0.2878
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 141ms/step - accuracy: 0.9418 - loss: 0.1608 - val_accuracy: 0.8784 - val_loss: 0.3082
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 124ms/step - accuracy: 0.9472 - loss: 0.1410 - val_accuracy: 0.8835 - val_loss: 0.3076
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 119ms/step - accuracy: 0.9667 - loss: 0.1031 - val_accuracy: 0.8836 - val_loss: 0.3570
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 138ms/step - accuracy: 0.9780 - loss: 0.0740 - val_accuracy: 0.8822 - val_loss: 0.3825
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 108ms/step - accuracy: 0.9853 - loss: 0.0541 - val_accuracy: 0.8826 - val_loss: 0.3927
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 124ms/step - accuracy: 0.9978 - loss: 0.0122 - val_accuracy: 0.8549 - val_loss: 0.5857
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 127ms/step - accuracy: 0.9957 - loss: 0.0167 - val_accuracy: 0.8812 - val_loss: 0.4896
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 138ms/step - accuracy: 0.9999 - loss: 0.0026 - val_accuracy: 0.8812 - val_loss: 0.5739
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 122ms/step - accuracy: 0.9831 - loss: 0.0956 - val_accuracy: 0.8790 - val_loss: 0.4887
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 104ms/step - accuracy: 1.0000 - loss: 0.0023 - val_accuracy: 0.8809 - val_loss: 0.5656
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 124ms/step - accuracy: 1.0000 - loss: 9.2220e-04 - val_accuracy: 0.8800 - val_loss: 0.6525
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 105ms/step - accuracy: 1.0000 - loss: 4.2768e-04 - val_accuracy: 0.8802 - val_loss: 0.7260
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 7s 164ms/step - accuracy: 1.0000 - loss: 2.2328e-04 - val_accuracy: 0.8801 - val_loss: 0.7734
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 106ms/step - accuracy: 1.0000 - loss: 1.3643e-04 - val_accuracy: 0.8802 - val_loss: 0.8075
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 122ms/step - accuracy: 1.0000 - loss: 9.4768e-05 - val_accuracy: 0.8795 - val_loss: 0.8379
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 134ms/step - accuracy: 1.0000 - loss: 7.3872e-05 - val_accuracy: 0.8798 - val_loss: 0.8558
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 98ms/step - accuracy: 1.0000 - loss: 6.3061e-05 - val_accuracy: 0.8798 - val_loss: 0.8741
```

# Extracting and displaying the keys of the training history for the model with 128 units.

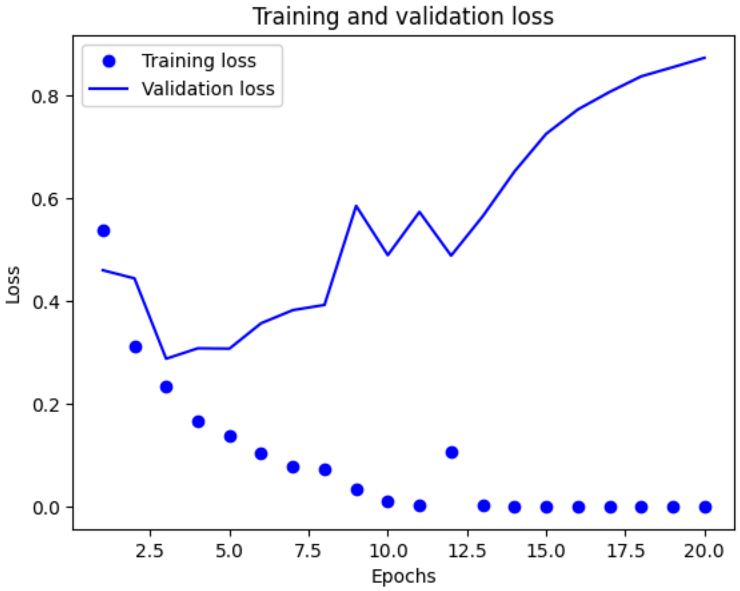
```
history_dict_128_3 = history_128_3.history
history_dict_128_3.keys()
```

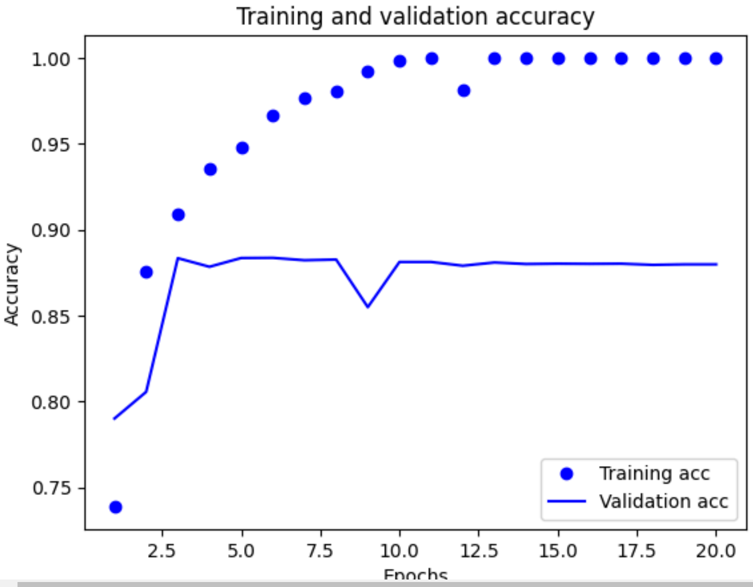
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

# Plotting training and validation loss, followed by training and validation accuracy for the model with 128 units.

```
loss_value128_3 = history_dict_128_3["loss"]
val_loss_value128_3 = history_dict_128_3["val_loss"]
```

```
epochs_128 = range(1, len(loss_value128_3) + 1)
plot647.plot(epochs_128, loss_value128_3, "bo", label="Training loss")
plot647.plot(epochs_128, val_loss_value128_3, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()


plot647.clf()
accuracy_128 = history_dict_128_3["accuracy"]
val_accuracy_128 = history_dict_128_3["val_accuracy"]
plot647.plot(epochs_128, accuracy_128, "bo", label="Training acc")
plot647.plot(epochs_128, val_accuracy_128, "b", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





# Training the model with 128 units for 2 epochs and evaluating its performance on the test set.

```
history_128_3 = model_128units_647.fit(x_train, y_train, epochs=2, batch_size=512)
results_128_units_3 = model_128units_647.evaluate(x_test, y_test)
results_128_units_3
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 77ms/step - accuracy: 0.9251 - loss: 0.4173
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 5s 84ms/step - accuracy: 0.9713 - loss: 0.0868
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8740 - loss: 0.3972
[0.39473628997802734, 0.8768399953842163]
```

```
model_128units_647.predict(x_test) # Predicting the output for the test data using the trained model with 128 units.
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step
array([[0.00944366],
       [1.        ],
       [0.7818167 ],
       ...,
       [0.01277798],
       [0.00162015],
       [0.9119414 ]], dtype=float32)
```

MSE Loss Function model with 16 units and 3-layers

# Defining, compiling, and training a model with 16 units using Mean Squared Error (MSE) as the loss function for 20 epochs.

```
MSE_model_16_647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
# compilation of model
MSE_model_16_647.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
# validation of model
x_val_MSE_16 = x_train[:10000]
partial_x_train_16 = x_train[10000:]

y_val_MSE_16 = y_train[:10000]
partial_y_train_16 = y_train[10000:]
# Model Fit

history_MSE_647 = MSE_model_16_647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val_MSE_16, y_val_MSE_16))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 77ms/step - accuracy: 0.6727 - loss: 0.2204 - val_accuracy: 0.8594 - val_loss: 0.1409
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.8798 - loss: 0.1214 - val_accuracy: 0.8838 - val_loss: 0.1024
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9118 - loss: 0.0834 - val_accuracy: 0.8684 - val_loss: 0.0993
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9276 - loss: 0.0648 - val_accuracy: 0.8735 - val_loss: 0.0936
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9443 - loss: 0.0521 - val_accuracy: 0.8875 - val_loss: 0.0840
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9546 - loss: 0.0441 - val_accuracy: 0.8877 - val_loss: 0.0825
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9646 - loss: 0.0371 - val_accuracy: 0.8859 - val_loss: 0.0835
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9690 - loss: 0.0318 - val_accuracy: 0.8847 - val_loss: 0.0851
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9763 - loss: 0.0272 - val_accuracy: 0.8821 - val_loss: 0.0877
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.9772 - loss: 0.0251 - val_accuracy: 0.8779 - val_loss: 0.0897
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - accuracy: 0.9811 - loss: 0.0214 - val_accuracy: 0.8791 - val_loss: 0.0901
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9840 - loss: 0.0194 - val_accuracy: 0.8805 - val_loss: 0.0906
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9869 - loss: 0.0160 - val_accuracy: 0.8755 - val_loss: 0.0960
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9880 - loss: 0.0154 - val_accuracy: 0.8783 - val_loss: 0.0946
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9882 - loss: 0.0139 - val_accuracy: 0.8773 - val_loss: 0.0954
Epoch 16/20
```

```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9917 - loss: 0.0101 - val_accuracy: 0.8756 - val_loss: 0.0969
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9911 - loss: 0.0105 - val_accuracy: 0.8743 - val_loss: 0.0984
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9945 - loss: 0.0073 - val_accuracy: 0.8724 - val_loss: 0.1001
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9888 - loss: 0.0114 - val_accuracy: 0.8731 - val_loss: 0.1004
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9928 - loss: 0.0080 - val_accuracy: 0.8728 - val_loss: 0.1012
```

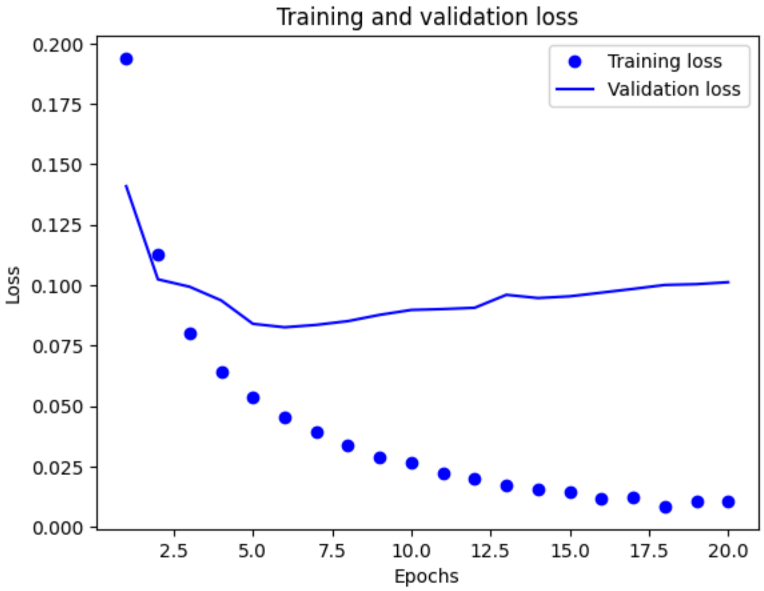
# Extracting and displaying the keys from the history dictionary of the MSE model.

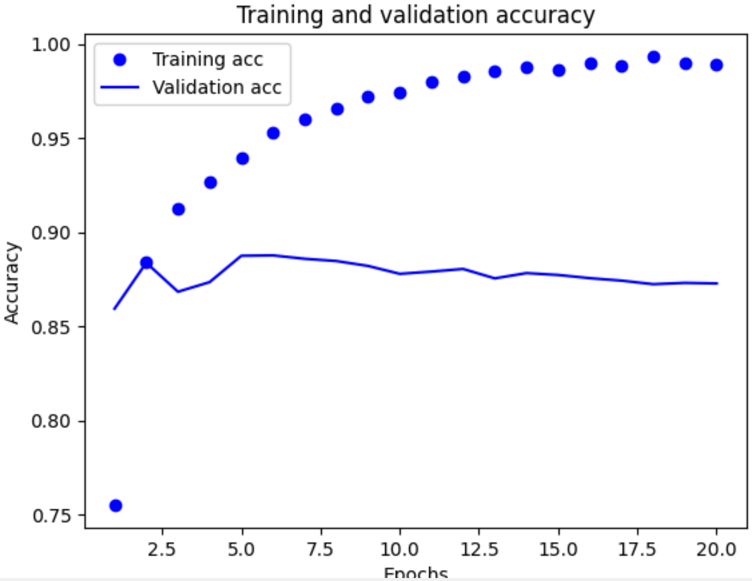
```
historydict_MSE_647 = history_MSE_647.history
historydict_MSE_647.keys()
```

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

# Plotting the training and validation loss, followed by training and validation accuracy for the MSE model.

```
import matplotlib.pyplot as plot647
loss_value_MSE_16_3 = historydict_MSE_647["loss"]
val_loss_value_MSE_16_3 = historydict_MSE_647["val_loss"]
epochs_MSE = range(1, len(loss_value_MSE_16_3) + 1)
plot647.plot(epochs_MSE, loss_value_MSE_16_3, "bo", label="Training loss")
plot647.plot(epochs_MSE, val_loss_value_MSE_16_3, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()


plot647.clf()
acc_MSE = historydict_MSE_647["accuracy"]
val_acc_MSE = historydict_MSE_647["val_accuracy"]
plot647.plot(epochs_MSE, acc_MSE, "bo", label="Training acc")
plot647.plot(epochs_MSE, val_acc_MSE, "b", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





# Training the MSE model for 8 epochs and evaluating its performance on the test set.

```
MSE_model_16_647.fit(x_train, y_train, epochs=8, batch_size=512)
results_MSE_647 = MSE_model_16_647.evaluate(x_test, y_test)
results_MSE_647
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9467 - loss: 0.0442
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9621 - loss: 0.0342
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9695 - loss: 0.0292
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9721 - loss: 0.0269
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.9761 - loss: 0.0237
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9813 - loss: 0.0191
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9806 - loss: 0.0195
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9821 - loss: 0.0177
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8622 - loss: 0.1141
[0.11139561235904694, 0.8662800192832947]
```

```
MSE_model_16_647.predict(x_test) # Predicting the output using the trained MSE model on the test data.
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.01404752],
       [0.99999624],
       [0.78736985],
       ...,
       [0.17270125],
       [0.00390387],
       [0.7782587 ]], dtype=float32)
```

# Defining and training a neural network model using the 'tanh' activation function and 'mse' loss.

```
tanh_647 = keras.Sequential([
    layers.Dense(16, activation="tanh"),
    layers.Dense(1, activation="sigmoid")
])

tanh_647.compile(optimizer='rmsprop',
              loss='mse',
              metrics=['accuracy'])

x_val_tanh = x_train[:10000]
partial_x_train = x_train[10000:]

y_val_tanh = y_train[:10000]
partial_y_train = y_train[10000:]


historytanh_model = tanh_647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val_tanh, y_val_tanh))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 67ms/step - accuracy: 0.7109 - loss: 0.2014 - val_accuracy: 0.8227 - val_loss: 0.1441
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.8798 - loss: 0.1181 - val_accuracy: 0.8703 - val_loss: 0.1108
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9083 - loss: 0.0905 - val_accuracy: 0.8829 - val_loss: 0.0962
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9239 - loss: 0.0737 - val_accuracy: 0.8793 - val_loss: 0.0934
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.9291 - loss: 0.0674 - val_accuracy: 0.8821 - val_loss: 0.0891
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9378 - loss: 0.0583 - val_accuracy: 0.8759 - val_loss: 0.0930
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9419 - loss: 0.0539 - val_accuracy: 0.8863 - val_loss: 0.0852
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9486 - loss: 0.0488 - val_accuracy: 0.8860 - val_loss: 0.0837
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9572 - loss: 0.0432 - val_accuracy: 0.8809 - val_loss: 0.0851
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9597 - loss: 0.0407 - val_accuracy: 0.8834 - val_loss: 0.0860
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9686 - loss: 0.0345 - val_accuracy: 0.8818 - val_loss: 0.0844
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9666 - loss: 0.0353 - val_accuracy: 0.8736 - val_loss: 0.0897
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9697 - loss: 0.0310 - val_accuracy: 0.8814 - val_loss: 0.0877
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9757 - loss: 0.0278 - val_accuracy: 0.8799 - val_loss: 0.0869
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 54ms/step - accuracy: 0.9800 - loss: 0.0249 - val_accuracy: 0.8791 - val_loss: 0.0878
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9833 - loss: 0.0228 - val_accuracy: 0.8748 - val_loss: 0.0910
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9802 - loss: 0.0239 - val_accuracy: 0.8789 - val_loss: 0.0917
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9844 - loss: 0.0204 - val_accuracy: 0.8735 - val_loss: 0.0925
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9822 - loss: 0.0215 - val_accuracy: 0.8756 - val_loss: 0.0924
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9863 - loss: 0.0187 - val_accuracy: 0.8760 - val_loss: 0.0953
```

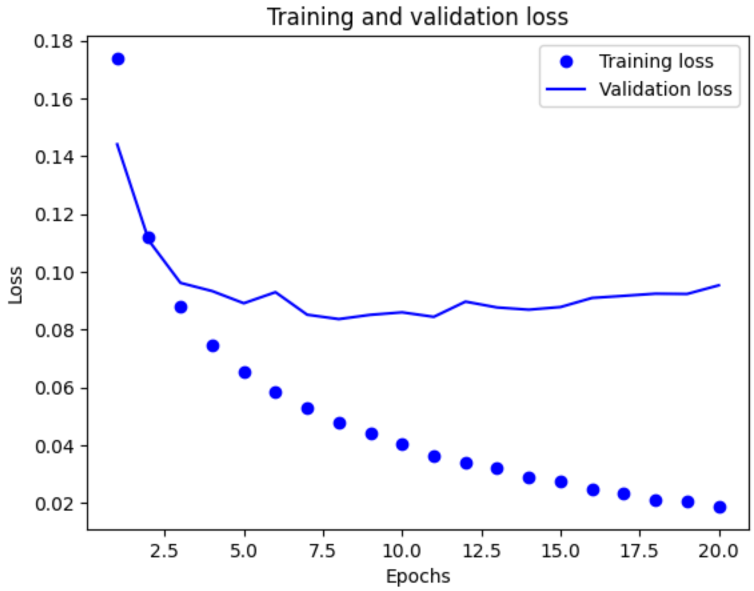
# Retrieving and displaying the keys of the history dictionary for the tanh model.

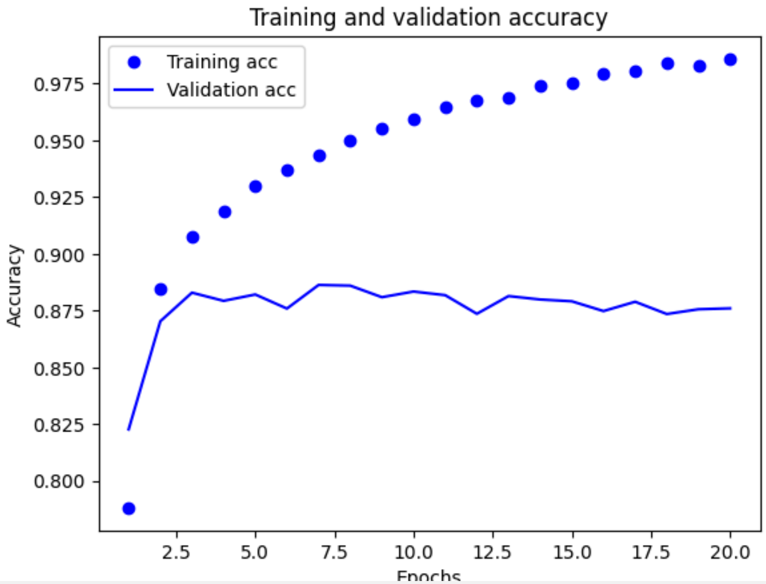
```
historydict_tanh_647 = historytanh_model.history
historydict_tanh_647.keys()
```

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

# Plotting the training and validation loss, and accuracy for the tanh model over epochs.

```
loss_value_tanh_647= historydict_tanh_647["loss"]
val_loss_value_tanh_647 = historydict_tanh_647["val_loss"]
epochs_tanh = range(1, len(loss_value_tanh_647) + 1)
plot647.plot(epochs_tanh, loss_value_tanh_647, "bo", label="Training loss")
plot647.plot(epochs_tanh, val_loss_value_tanh_647, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()

plot647.clf()
acc_tanh = historydict_tanh_647["accuracy"]
val_acc_tanh = historydict_tanh_647["val_accuracy"]
plot647.plot(epochs_tanh, acc_tanh, "bo", label="Training acc")
plot647.plot(epochs_tanh, val_acc_tanh, "b", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





# Training the tanh model for 8 epochs and evaluating it on the test set.

```
tanh_647.fit(x_train, y_train, epochs=8, batch_size=512)
results_tanh_647 = tanh_647.evaluate(x_test, y_test)
results_tanh_647
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9405 - loss: 0.0490
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 38ms/step - accuracy: 0.9537 - loss: 0.0409
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9601 - loss: 0.0366
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - accuracy: 0.9637 - loss: 0.0342
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9675 - loss: 0.0316
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9688 - loss: 0.0302
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.9739 - loss: 0.0269
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9751 - loss: 0.0260
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8624 - loss: 0.1084
[0.1056099534034729, 0.8668799996376038]
```

Adam Operator with 16 units and 3-layers

# Defining, compiling, and training a model with the Adam optimizer for 20 epochs.

```
adam_647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
     layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])

adam_647.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

x_adam_647 = x_train[:10000]
partial_x_train_16 = x_train[10000:]

y_adam_647 = y_train[:10000]
partial_y_train_16 = y_train[10000:]


historyadam_647 = adam_647.fit(partial_x_train_16,
                    partial_y_train_16,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_adam_647, y_adam_647))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 80ms/step - accuracy: 0.6794 - loss: 0.6317 - val_accuracy: 0.8581 - val_loss: 0.3981
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9003 - loss: 0.3120 - val_accuracy: 0.8900 - val_loss: 0.2799
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9417 - loss: 0.1794 - val_accuracy: 0.8854 - val_loss: 0.2906
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9656 - loss: 0.1179 - val_accuracy: 0.8835 - val_loss: 0.3093
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9791 - loss: 0.0832 - val_accuracy: 0.8810 - val_loss: 0.3485
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9888 - loss: 0.0551 - val_accuracy: 0.8769 - val_loss: 0.3957
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9951 - loss: 0.0349 - val_accuracy: 0.8742 - val_loss: 0.4427
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9970 - loss: 0.0223 - val_accuracy: 0.8734 - val_loss: 0.4898
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9992 - loss: 0.0135 - val_accuracy: 0.8711 - val_loss: 0.5323
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9997 - loss: 0.0094 - val_accuracy: 0.8700 - val_loss: 0.5721
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9999 - loss: 0.0061 - val_accuracy: 0.8694 - val_loss: 0.6104
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 1.0000 - loss: 0.0041 - val_accuracy: 0.8691 - val_loss: 0.6482
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 0.8681 - val_loss: 0.6836
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 1.0000 - loss: 0.0021 - val_accuracy: 0.8678 - val_loss: 0.7159
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 1.0000 - loss: 0.0015 - val_accuracy: 0.8678 - val_loss: 0.7465
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 0.8672 - val_loss: 0.7740
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 1.0000 - loss: 8.8557e-04 - val_accuracy: 0.8666 - val_loss: 0.7975
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 1.0000 - loss: 7.5288e-04 - val_accuracy: 0.8666 - val_loss: 0.8194
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 1.0000 - loss: 6.1903e-04 - val_accuracy: 0.8666 - val_loss: 0.8401
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 1.0000 - loss: 5.0481e-04 - val_accuracy: 0.8669 - val_loss: 0.8583
```

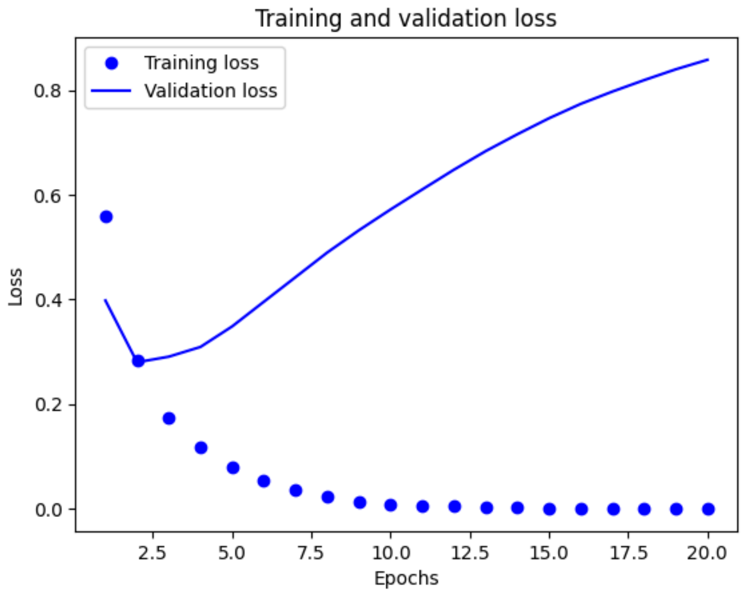
# Extracting and displaying the keys from the training history of the Adam optimizer model.

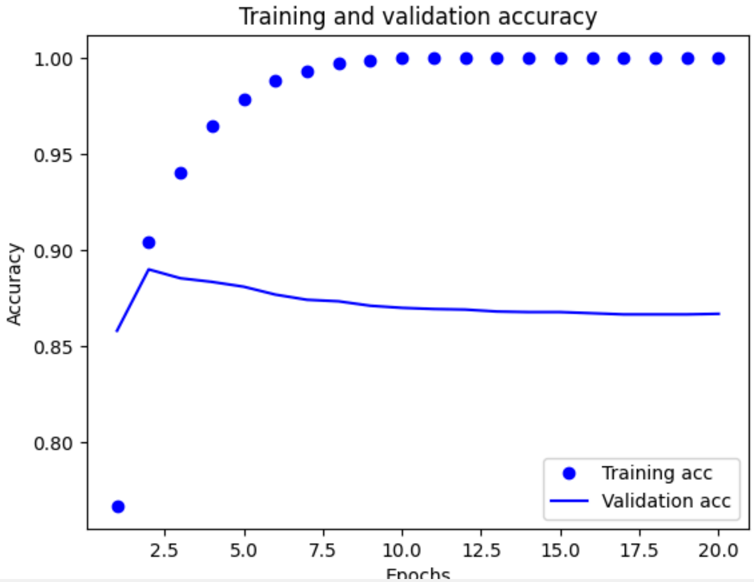
```
historydict_adam_647 = historyadam_647.history
historydict_adam_647.keys()
```

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

# Plotting the training and validation loss and accuracy for the Adam optimizer model.

```
loss_value_adam_647 = historydict_adam_647["loss"]
val_loss_value_adam_647 = historydict_adam_647["val_loss"]
epochs_adam = range(1, len(loss_value_adam_647) + 1)
plot647.plot(epochs_adam, loss_value_adam_647, "bo", label="Training loss")
plot647.plot(epochs_adam, val_loss_value_adam_647, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()


plot647.clf()
acc_adam = historydict_adam_647["accuracy"]
val_acc_adam = historydict_adam_647["val_accuracy"]
plot647.plot(epochs_adam, acc_adam, "bo", label="Training acc")
plot647.plot(epochs_adam, val_acc_adam, "b", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```


Training and validation loss


Training and validation accuracy

# Training the Adam optimizer model for 4 epochs and evaluating its performance on the test set.

```
adam_647.fit(x_train, y_train, epochs=4, batch_size=512)
results_adam = adam_647.evaluate(x_test, y_test)
```

```
results_adam
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9371 - loss: 0.2947
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9675 - loss: 0.1098
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9852 - loss: 0.0613
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.9921 - loss: 0.0405
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8563 - loss: 0.5531
[0.5533235669136047, 0.8574399948120117]
```

Regularization model with 16 units and 2-layers

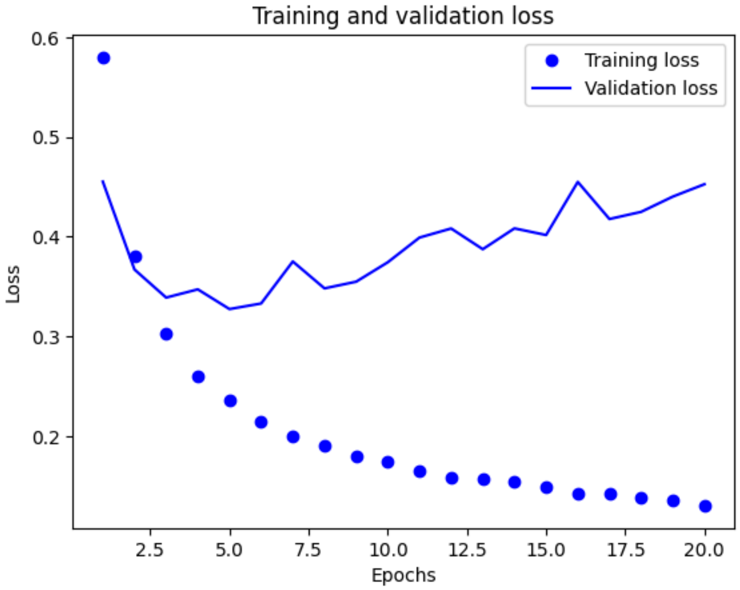
# Defining and training a model with L2 regularization on the dense layers and evaluating its performance.

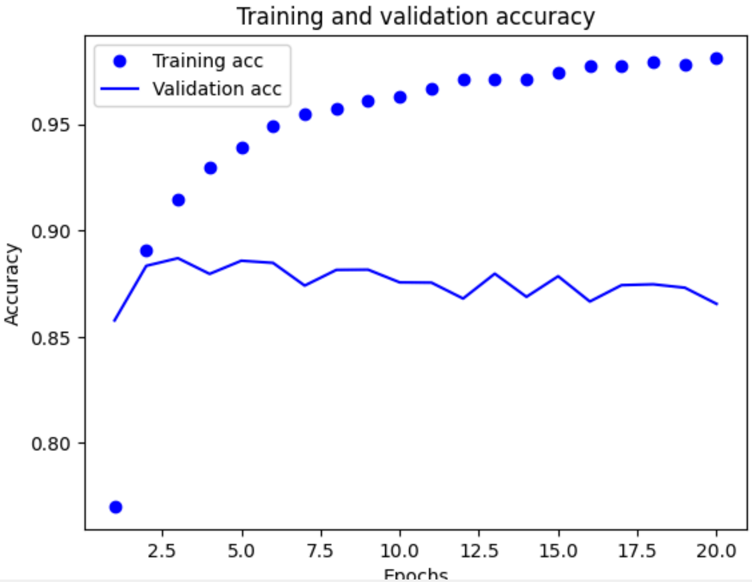
```
from tensorflow.keras import regularizers
regularization647 = keras.Sequential([
    layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(16, activation="relu",kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
regularization647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history_regularization647 = regularization647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
historydict_regularization647 = history_regularization647.history
historydict_regularization647.keys()
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 67ms/step - accuracy: 0.6849 - loss: 0.6530 - val_accuracy: 0.8577 - val_loss: 0.4552
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.8889 - loss: 0.3976 - val_accuracy: 0.8834 - val_loss: 0.3671
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9169 - loss: 0.3073 - val_accuracy: 0.8870 - val_loss: 0.3390
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9319 - loss: 0.2634 - val_accuracy: 0.8796 - val_loss: 0.3472
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9457 - loss: 0.2274 - val_accuracy: 0.8858 - val_loss: 0.3275
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9543 - loss: 0.2036 - val_accuracy: 0.8848 - val_loss: 0.3331
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9623 - loss: 0.1906 - val_accuracy: 0.8741 - val_loss: 0.3752
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9581 - loss: 0.1890 - val_accuracy: 0.8815 - val_loss: 0.3482
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9662 - loss: 0.1725 - val_accuracy: 0.8816 - val_loss: 0.3549
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9692 - loss: 0.1658 - val_accuracy: 0.8756 - val_loss: 0.3744
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9710 - loss: 0.1577 - val_accuracy: 0.8755 - val_loss: 0.3991
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9717 - loss: 0.1563 - val_accuracy: 0.8680 - val_loss: 0.4082
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9726 - loss: 0.1556 - val_accuracy: 0.8797 - val_loss: 0.3875
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9768 - loss: 0.1460 - val_accuracy: 0.8688 - val_loss: 0.4084
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9779 - loss: 0.1418 - val_accuracy: 0.8785 - val_loss: 0.4017
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - accuracy: 0.9827 - loss: 0.1351 - val_accuracy: 0.8666 - val_loss: 0.4549
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9809 - loss: 0.1354 - val_accuracy: 0.8743 - val_loss: 0.4177
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9836 - loss: 0.1315 - val_accuracy: 0.8747 - val_loss: 0.4249
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9857 - loss: 0.1256 - val_accuracy: 0.8731 - val_loss: 0.4402
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9862 - loss: 0.1231 - val_accuracy: 0.8655 - val_loss: 0.4526
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

# Plotting training and validation loss and accuracy for the model with L2 regularization.

```
loss_valu_647 = historydict_regularization647["loss"]
val_loss_value_r_647 = historydict_regularization647["val_loss"]
epochs_r = range(1, len(loss_valu_647) + 1)
plot647.plot(epochs_r, loss_valu_647, "bo", label="Training loss")
plot647.plot(epochs_r, val_loss_value_r_647, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()

plot647.clf()
acc_r = historydict_regularization647["accuracy"]
val_acc_r = historydict_regularization647["val_accuracy"]
plot647.plot(epochs_r, acc_r, "bo", label="Training acc")
plot647.plot(epochs_r, val_acc_r, "b", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```





# Training and evaluating the model with L2 regularization for 8 epochs.

```
regularization647.fit(x_train, y_train, epochs=8, batch_size=512)
results_regularization_647 = regularization647.evaluate(x_test, y_test)
```

```
results_regularization_647
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9396 - loss: 0.2528
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.9524 - loss: 0.1998
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9594 - loss: 0.1829
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9640 - loss: 0.1708
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.9646 - loss: 0.1662
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9671 - loss: 0.1619
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9655 - loss: 0.1628
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9710 - loss: 0.1540
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8642 - loss: 0.4343
[0.4305916130542755, 0.8685600161552429]
```

Dropout function with 16 units and 3-layers

# Defining a model with dropout layers, compiling, and fitting it for 20 epochs with validation.

```
from tensorflow.keras import regularizers
Dropout647 = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")
])
Dropout647.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])

history_Dropout_647 = Dropout647.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
historydict_Dropout_647 = history_Dropout_647.history
historydict_Dropout_647.keys()
```

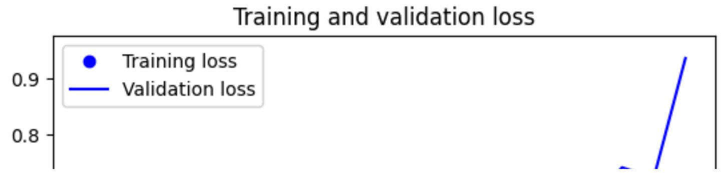
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.5631 - loss: 0.6837 - val_accuracy: 0.7499 - val_loss: 0.6256
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.7289 - loss: 0.6129 - val_accuracy: 0.8350 - val_loss: 0.5389
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.8147 - loss: 0.5388 - val_accuracy: 0.8315 - val_loss: 0.4690
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.8528 - loss: 0.4679 - val_accuracy: 0.8631 - val_loss: 0.4463
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 59ms/step - accuracy: 0.8813 - loss: 0.4261 - val_accuracy: 0.8733 - val_loss: 0.3941
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.8881 - loss: 0.3854 - val_accuracy: 0.8587 - val_loss: 0.4154
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9014 - loss: 0.3414 - val_accuracy: 0.8691 - val_loss: 0.3662
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9149 - loss: 0.3038 - val_accuracy: 0.8698 - val_loss: 0.3733
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9202 - loss: 0.2801 - val_accuracy: 0.8694 - val_loss: 0.3990
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9353 - loss: 0.2419 - val_accuracy: 0.8703 - val_loss: 0.4136
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9366 - loss: 0.2224 - val_accuracy: 0.8693 - val_loss: 0.4385
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9482 - loss: 0.2018 - val_accuracy: 0.8654 - val_loss: 0.4879
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9494 - loss: 0.1934 - val_accuracy: 0.8662 - val_loss: 0.4946
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.9565 - loss: 0.1706 - val_accuracy: 0.8646 - val_loss: 0.5467
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9556 - loss: 0.1647 - val_accuracy: 0.8673 - val_loss: 0.5990
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9613 - loss: 0.1492 - val_accuracy: 0.8671 - val_loss: 0.6019
```

```
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9613 - loss: 0.1520 - val_accuracy: 0.8646 - val_loss: 0.6729
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9657 - loss: 0.1407 - val_accuracy: 0.8625 - val_loss: 0.7416
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9671 - loss: 0.1351 - val_accuracy: 0.8651 - val_loss: 0.7225
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9675 - loss: 0.1338 - val_accuracy: 0.8545 - val_loss: 0.9370
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

# Plotting training and validation loss, and accuracy for the model with dropout layers.

```
loss_val_647 = historydict_Dropout_647["loss"]
val_loss_val_d_647 = historydict_Dropout_647["val_loss"]
epochs_d = range(1, len(loss_val_647) + 1)
plot647.plot(epochs_d, loss_val_647, "bo", label="Training loss")
plot647.plot(epochs_d, val_loss_val_d_647, "b", label="Validation loss")
plot647.title("Training and validation loss")
plot647.xlabel("Epochs")
plot647.ylabel("Loss")
plot647.legend()
plot647.show()
```

```
plot647.clf()
acc_d = historydict_Dropout_647["accuracy"]
val_acc_d = historydict_Dropout_647["val_accuracy"]
plot647.plot(epochs_d, acc_d, "bo", label="Training acc")
plot647.plot(epochs_d, val_acc_d, "b", label="Validation acc")
plot647.title("Training and validation accuracy")
plot647.xlabel("Epochs")
plot647.ylabel("Accuracy")
plot647.legend()
plot647.show()
```



# Fitting the Dropout model for 8 epochs and evaluating it on the test data.

```
Dropout647.fit(x_train, y_train, epochs=8, batch_size=512)
results_Dropout647 = Dropout647.evaluate(x_test, y_test)
results_Dropout647
```

```
Epoch 1/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9101 - loss: 0.3866
Epoch 2/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9152 - loss: 0.3057
Epoch 3/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9230 - loss: 0.2645
Epoch 4/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9302 - loss: 0.2363
Epoch 5/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9373 - loss: 0.2198
Epoch 6/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9364 - loss: 0.2137
Epoch 7/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9381 - loss: 0.2124
Epoch 8/8
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9435 - loss: 0.1974
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8585 - loss: 0.5623
[0.5447589755058289, 0.8576800227165222]
```

Training model with hyper tuned parameters with 32 units and 3 -layers

# Defining and training a model with L2 regularization and dropout layers, using RMSprop optimizer and MSE loss function.