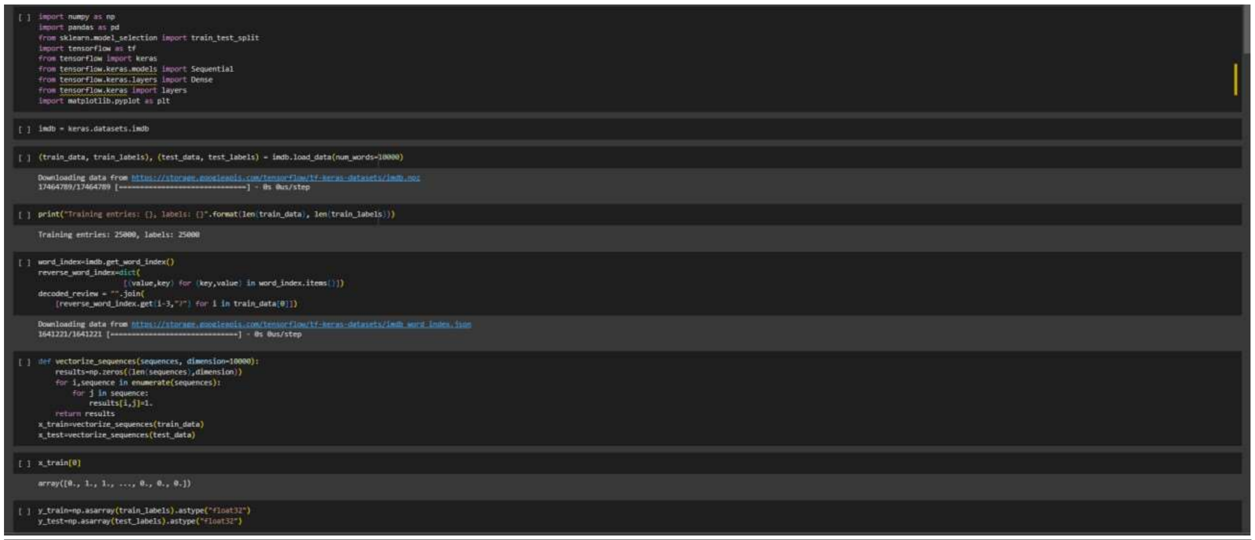
AML -- Assignment 1: Neural Networks

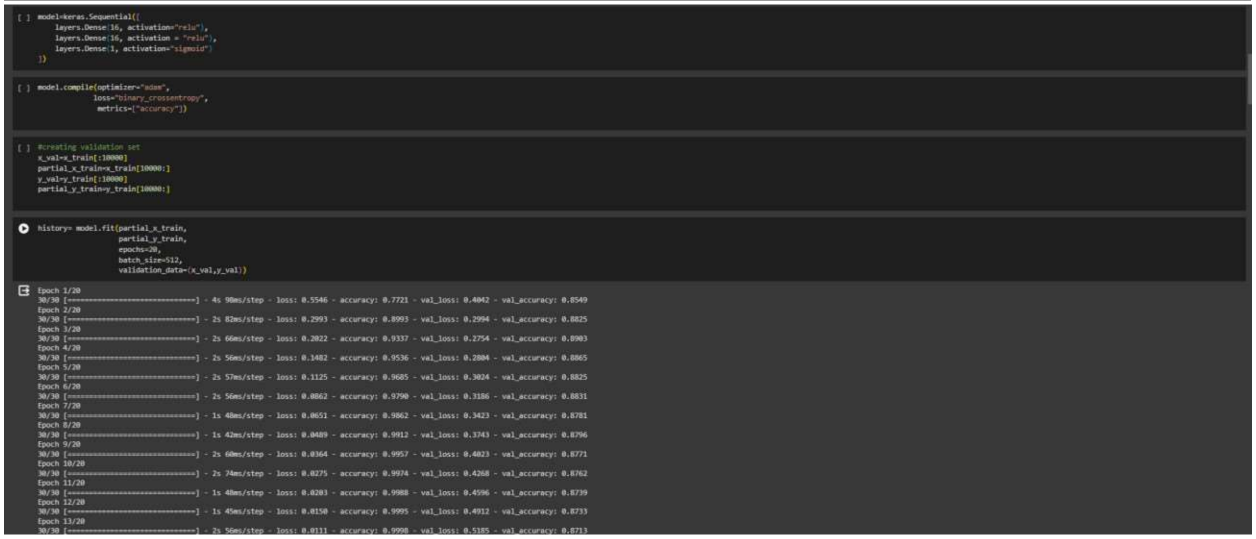
Using TensorFlow and Keras, this code builds a neural network to categorize movie reviews as good or bad based on IMDB data. It begins by loading the IMDB dataset Keras gave, which consists of preprocessed movie reviews in the form of sequences of numbers, with each integer representing a word from a dictionary of the 10,000 most commonly occurring terms. The method then vectorizes these sequences, transforming them into a binary matrix where each row represents a review, and each column represents a dictionary term. The presence of a term in a review is indicated by setting the matrix entry to 1. The labels (positive and negative sentiment) have also been prepared for training and testing.

A basic neural network model is then defined using Keras' Sequential API. It is made up of three dense (completely associated) layers, each having 16 units and ReLU activation functions, except the output layer, which has a single unit and a sigmoid activation function for binary classification. The model is constructed using the Adam optimizer and the binary cross-entropy loss function, with accuracy as the metric. To avoid overfitting, a validation set is formed by dividing a portion of the training data. The model is trained on training data and verified on validation data for 20 epochs using a batch size of 512. Following training, the model is evaluated against test data, and its performance metrics (loss and accuracy) are printed. The test accuracy is found to be approximately 85.58%. Finally, the model is used to predict the sentiment of the test data.

Code:



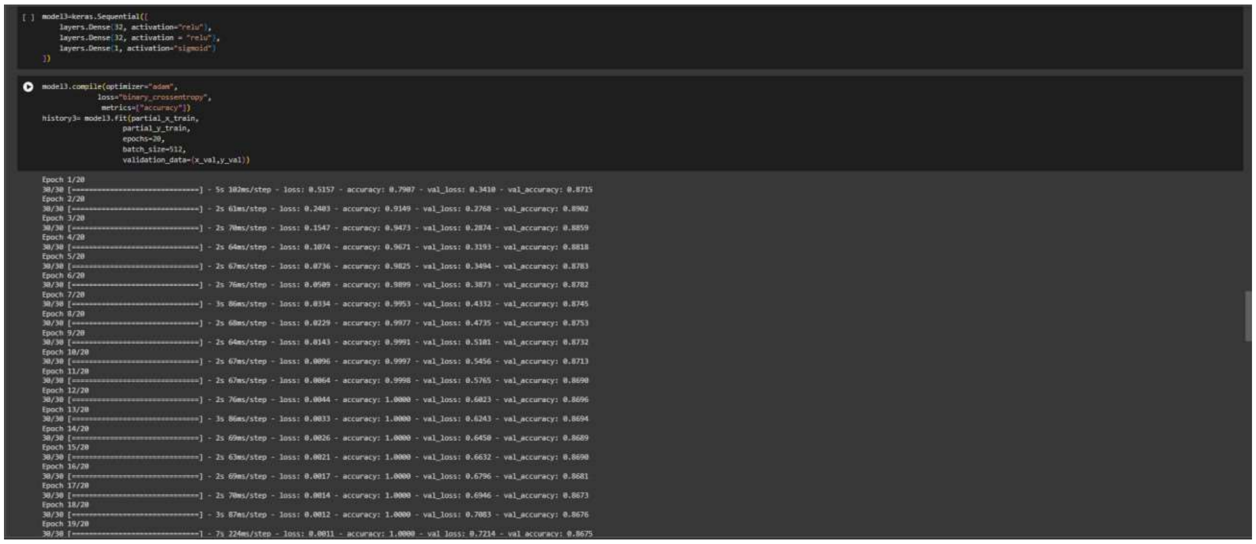
```
[ ] import numpy as np
    import pandas as pd
    from sklearn.model_selection import train_test_split
    import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense
    from tensorflow.keras import layers
    import matplotlib.pyplot as plt

[ ] imdb = keras.datasets.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

[ ] print("Training entries: {}, labels: {}".format(len(train_data), len(train_labels)))

    Training entries: 25000, labels: 25000

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

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

[ ] 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)

[ ] x_train[0]

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

[ ] y_train=np.asarray(train_labels).astype("float32")
    y_test=np.asarray(test_labels).astype("float32")
```



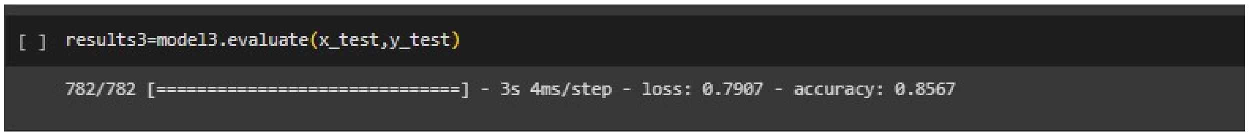
```
[ ] model=keras.Sequential([
        layers.Dense(16, activation="relu"),
        layers.Dense(16, activation = "relu"),
        layers.Dense(1, activation="sigmoid")
    ])

[ ] model.compile(optimizer="adam",
                 loss="binary_crossentropy",
                  metrics=["accuracy"])

[ ] #creating validation set
    x_val=x_train[:10000]
    partial_x_train=x_train[10000:]
    y_val=y_train[:10000]
    partial_y_train=y_train[10000:]

    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 98ms/step - loss: 0.5546 - accuracy: 0.7721 - val_loss: 0.4042 - val_accuracy: 0.8549
    Epoch 2/20
    30/30 [==============================] - 2s 82ms/step - loss: 0.2993 - accuracy: 0.8993 - val_loss: 0.2994 - val_accuracy: 0.8825
    Epoch 3/20
    30/30 [==============================] - 2s 66ms/step - loss: 0.2022 - accuracy: 0.9337 - val_loss: 0.2754 - val_accuracy: 0.8903
    Epoch 4/20
    30/30 [==============================] - 2s 56ms/step - loss: 0.1482 - accuracy: 0.9536 - val_loss: 0.2804 - val_accuracy: 0.8865
    Epoch 5/20
    30/30 [==============================] - 2s 57ms/step - loss: 0.1125 - accuracy: 0.9685 - val_loss: 0.3024 - val_accuracy: 0.8825
    Epoch 6/20
    30/30 [==============================] - 2s 56ms/step - loss: 0.0862 - accuracy: 0.9790 - val_loss: 0.3186 - val_accuracy: 0.8831
    Epoch 7/20
    30/30 [==============================] - 1s 48ms/step - loss: 0.0651 - accuracy: 0.9862 - val_loss: 0.3423 - val_accuracy: 0.8781
    Epoch 8/20
    30/30 [==============================] - 1s 42ms/step - loss: 0.0489 - accuracy: 0.9912 - val_loss: 0.3743 - val_accuracy: 0.8796
    Epoch 9/20
    30/30 [==============================] - 2s 60ms/step - loss: 0.0364 - accuracy: 0.9957 - val_loss: 0.4023 - val_accuracy: 0.8771
    Epoch 10/20
    30/30 [==============================] - 2s 74ms/step - loss: 0.0275 - accuracy: 0.9974 - val_loss: 0.4268 - val_accuracy: 0.8762
    Epoch 11/20
    30/30 [==============================] - 1s 48ms/step - loss: 0.0203 - accuracy: 0.9988 - val_loss: 0.4596 - val_accuracy: 0.8739
    Epoch 12/20
    30/30 [==============================] - 1s 45ms/step - loss: 0.0150 - accuracy: 0.9995 - val_loss: 0.4912 - val_accuracy: 0.8733
    Epoch 13/20
    30/30 [==============================] - 2s 56ms/step - loss: 0.0111 - accuracy: 0.9998 - val_loss: 0.5185 - val_accuracy: 0.8713
```

1. **You used two hidden layers. Try using one or three hidden layers and see how doing so affects validation and test accuracy.**

Code:



```
[ ] model3=keras.Sequential([
        layers.Dense(32, activation="relu"),
        layers.Dense(32, activation = "relu"),
        layers.Dense(1, activation="sigmoid")
    ])

    model3.compile(optimizer="adam",
                   loss="binary_crossentropy",
                    metrics=["accuracy"])
    history3= model3.fit(partial_x_train,
                         partial_y_train,
                         epochs=20,
                         batch_size=512,
                         validation_data=(x_val,y_val))

    Epoch 1/20
    30/30 [==============================] - 5s 102ms/step - loss: 0.5157 - accuracy: 0.7907 - val_loss: 0.3410 - val_accuracy: 0.8715
    Epoch 2/20
    30/30 [==============================] - 2s 61ms/step - loss: 0.2403 - accuracy: 0.9149 - val_loss: 0.2768 - val_accuracy: 0.8902
    Epoch 3/20
    30/30 [==============================] - 2s 70ms/step - loss: 0.1547 - accuracy: 0.9473 - val_loss: 0.2874 - val_accuracy: 0.8859
    Epoch 4/20
    30/30 [==============================] - 2s 64ms/step - loss: 0.1074 - accuracy: 0.9671 - val_loss: 0.3193 - val_accuracy: 0.8818
    Epoch 5/20
    30/30 [==============================] - 2s 67ms/step - loss: 0.0736 - accuracy: 0.9825 - val_loss: 0.3494 - val_accuracy: 0.8783
    Epoch 6/20
    30/30 [==============================] - 2s 76ms/step - loss: 0.0509 - accuracy: 0.9899 - val_loss: 0.3873 - val_accuracy: 0.8782
    Epoch 7/20
    30/30 [==============================] - 3s 86ms/step - loss: 0.0334 - accuracy: 0.9953 - val_loss: 0.4332 - val_accuracy: 0.8745
    Epoch 8/20
    30/30 [==============================] - 2s 68ms/step - loss: 0.0229 - accuracy: 0.9977 - val_loss: 0.4735 - val_accuracy: 0.8753
    Epoch 9/20
    30/30 [==============================] - 2s 64ms/step - loss: 0.0143 - accuracy: 0.9991 - val_loss: 0.5101 - val_accuracy: 0.8732
    Epoch 10/20
    30/30 [==============================] - 2s 67ms/step - loss: 0.0096 - accuracy: 0.9997 - val_loss: 0.5456 - val_accuracy: 0.8713
    Epoch 11/20
    30/30 [==============================] - 2s 67ms/step - loss: 0.0064 - accuracy: 0.9998 - val_loss: 0.5765 - val_accuracy: 0.8690
    Epoch 12/20
    30/30 [==============================] - 2s 76ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.6023 - val_accuracy: 0.8696
    Epoch 13/20
    30/30 [==============================] - 3s 86ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.6243 - val_accuracy: 0.8694
    Epoch 14/20
    30/30 [==============================] - 2s 69ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.6450 - val_accuracy: 0.8689
    Epoch 15/20
    30/30 [==============================] - 2s 63ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.6632 - val_accuracy: 0.8690
    Epoch 16/20
    30/30 [==============================] - 2s 69ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.6796 - val_accuracy: 0.8681
    Epoch 17/20
    30/30 [==============================] - 2s 70ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.6946 - val_accuracy: 0.8673
    Epoch 18/20
    30/30 [==============================] - 3s 87ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.7083 - val_accuracy: 0.8676
    Epoch 19/20
    30/30 [==============================] - 7s 224ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.7214 - val_accuracy: 0.8675
```



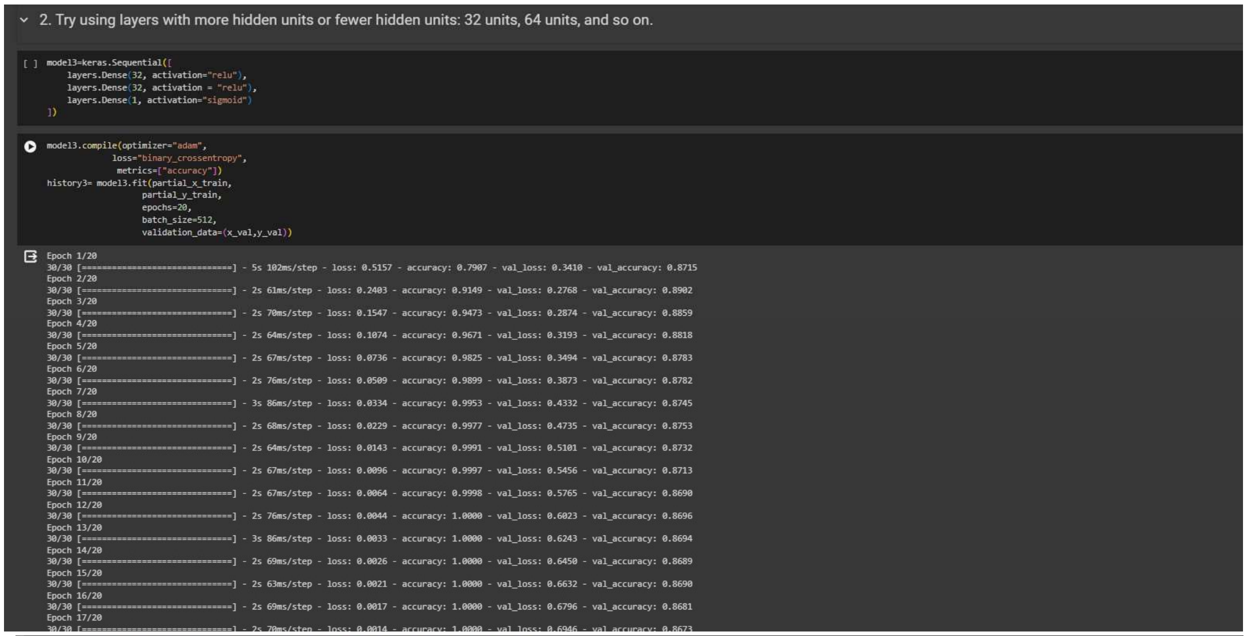
```
[ ] results3=model3.evaluate(x_test,y_test)

    782/782 [==============================] - 3s 4ms/step - loss: 0.7907 - accuracy: 0.8567
```

Two new models are defined and trained by the revised code. In contrast to the previous two new models, model 2 introduces a third hidden layer to the neural network. The design of Model 2 is comprised of three dense layers, a final output layer with a sigmoid activation function for binary classification, and each dense layer having 16 units with ReLU activation functions. In a similar manner, the binary cross-entropy loss function, accuracy as the metric, and the Adam optimizer are used in its construction. Next, this model is trained with a batch size of 512 across 20 epochs using the same training and validation data. Test data is used to evaluate it after training, and performance measures are computed.

When compared to the previous model, model2 has a test accuracy of about 85.90%, which is slightly higher than the initial model's test accuracy of about 85.58%. Interestingly, model2 has a higher training accuracy of 100% than the original model, which may indicate overfitting. However, the validation accuracy for model2 is comparable to the previous model, implying that the additional hidden layer may not significantly improve generalization performance. Overall, the main change in this code is the addition of an extra hidden layer to the neural network architecture, which results in a slight increase in test accuracy but no significant changes in validation accuracy.

2. **Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.**

Code:

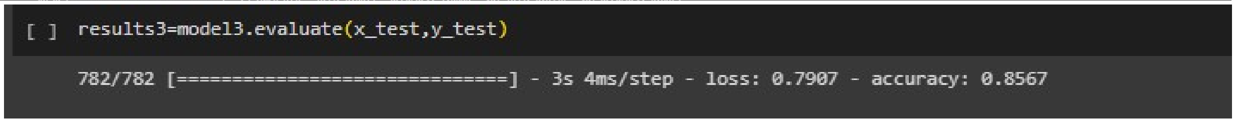


2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.

```
model3=keras.Sequential([
    layers.Dense(32, activation="relu"),
    layers.Dense(32, activation = "relu"),
    layers.Dense(1, activation="sigmoid")
])
```

```
model3.compile(optimizer="adam",
             loss="binary_crossentropy",
              metrics=["accuracy"])
history3= model3.fit(partial_x_train,
                   partial_y_train,
                   epochs=20,
                   batch_size=512,
                   validation_data=(x_val,y_val))
```

```
Epoch 1/20
30/30 [==============================] - 5s 102ms/step - loss: 0.5157 - accuracy: 0.7907 - val_loss: 0.3410 - val_accuracy: 0.8715
Epoch 2/20
30/30 [==============================] - 2s 61ms/step - loss: 0.2403 - accuracy: 0.9149 - val_loss: 0.2768 - val_accuracy: 0.8902
Epoch 3/20
30/30 [==============================] - 2s 70ms/step - loss: 0.1547 - accuracy: 0.9473 - val_loss: 0.2874 - val_accuracy: 0.8859
Epoch 4/20
30/30 [==============================] - 2s 64ms/step - loss: 0.1074 - accuracy: 0.9671 - val_loss: 0.3193 - val_accuracy: 0.8818
Epoch 5/20
30/30 [==============================] - 2s 67ms/step - loss: 0.0736 - accuracy: 0.9825 - val_loss: 0.3494 - val_accuracy: 0.8783
Epoch 6/20
30/30 [==============================] - 2s 76ms/step - loss: 0.0509 - accuracy: 0.9899 - val_loss: 0.3873 - val_accuracy: 0.8782
Epoch 7/20
30/30 [==============================] - 3s 86ms/step - loss: 0.0334 - accuracy: 0.9953 - val_loss: 0.4332 - val_accuracy: 0.8745
Epoch 8/20
30/30 [==============================] - 2s 68ms/step - loss: 0.0229 - accuracy: 0.9977 - val_loss: 0.4735 - val_accuracy: 0.8753
Epoch 9/20
30/30 [==============================] - 2s 64ms/step - loss: 0.0143 - accuracy: 0.9991 - val_loss: 0.5101 - val_accuracy: 0.8732
Epoch 10/20
30/30 [==============================] - 2s 67ms/step - loss: 0.0096 - accuracy: 0.9997 - val_loss: 0.5456 - val_accuracy: 0.8713
Epoch 11/20
30/30 [==============================] - 2s 67ms/step - loss: 0.0064 - accuracy: 0.9998 - val_loss: 0.5765 - val_accuracy: 0.8690
Epoch 12/20
30/30 [==============================] - 2s 76ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 0.6023 - val_accuracy: 0.8696
Epoch 13/20
30/30 [==============================] - 3s 86ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.6243 - val_accuracy: 0.8694
Epoch 14/20
30/30 [==============================] - 2s 69ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.6450 - val_accuracy: 0.8689
Epoch 15/20
30/30 [==============================] - 2s 63ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.6632 - val_accuracy: 0.8690
Epoch 16/20
30/30 [==============================] - 2s 69ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.6796 - val_accuracy: 0.8681
Epoch 17/20
30/30 [==============================] - 2s 70ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.6946 - val_accuracy: 0.8673
```



```
results3=model3.evaluate(x_test,y_test)
```

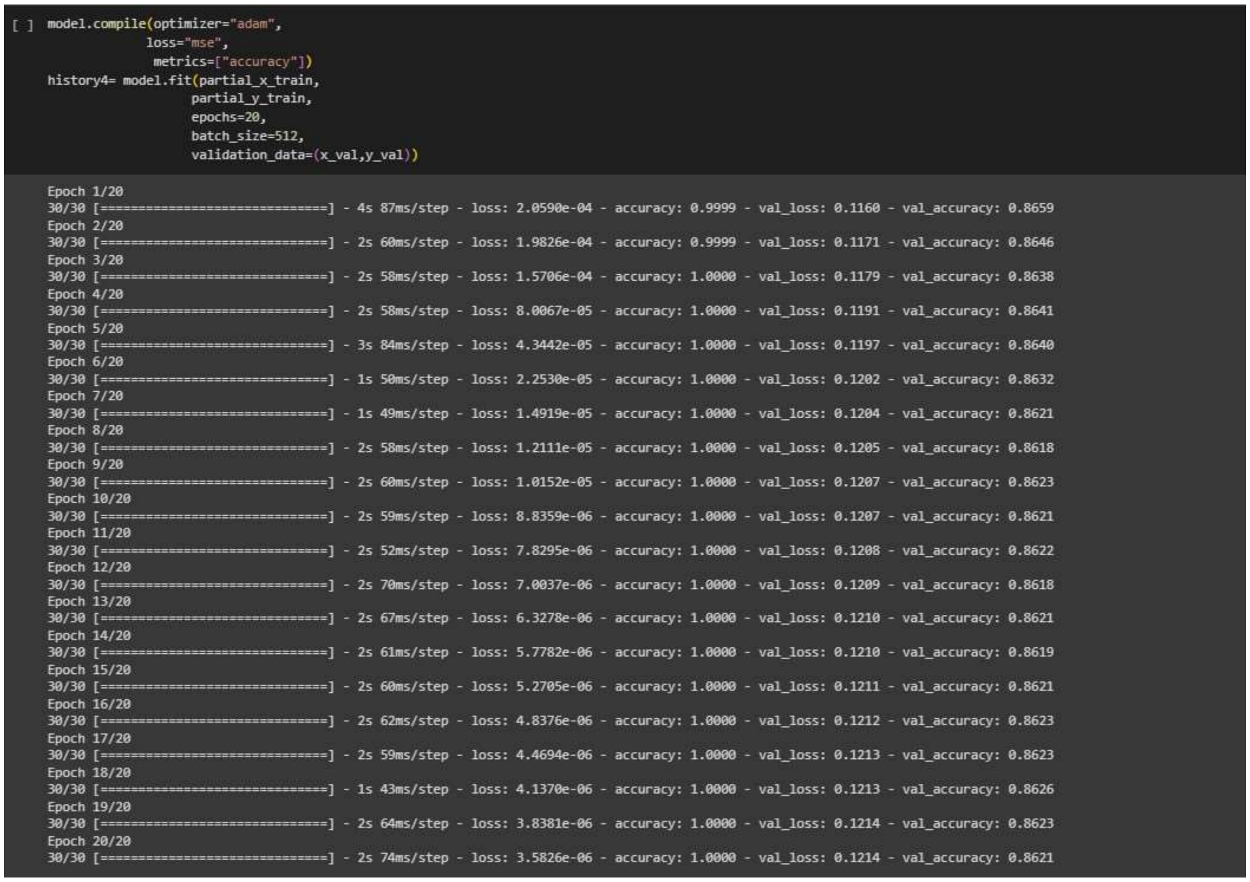
```
782/782 [==============================] - 3s 4ms/step - loss: 0.7907 - accuracy: 0.8567
```

The architecture of Model 3 differs from that of the first model. Model 3 consists of two hidden layers, each with 32 units and ReLU activation functions, and a final output layer that classifies binary data using a sigmoid activation function. The binary cross-entropy loss function, the Adam optimizer, and accuracy as the metric are used to construct the model. In a same manner, it is trained for 20 epochs with a batch size of 512 using the same training and validation data. Model3 is trained and assessed using test data, after which its performance indicators are computed.

The test accuracy of model 3 is roughly 85.67%, which is marginally less than the test accuracy of the initial model, which was approximately 85.58%. The accuracy difference is negligible, though. It's interesting to note that, like model2, model 3 has a greater training accuracy of 100%, suggesting the possibility of overfitting. Model 3's validation accuracy stays close to that of the original model, suggesting that the number of hidden units changed had no appreciable impact on generalization performance. The number of hidden units in the neural network design is the primary modification made to this code overall; this adjustment has a minor impact on test accuracy but has no appreciable effect on validation accuracy.

3. Try using the mse loss function instead of binary_crossentropy.

Code:



```
[ ] model.compile(optimizer="adam",
                  loss="mse",
                   metrics=["accuracy"])
    history4= model.fit(partial_x_train,
                        partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val,y_val))
```

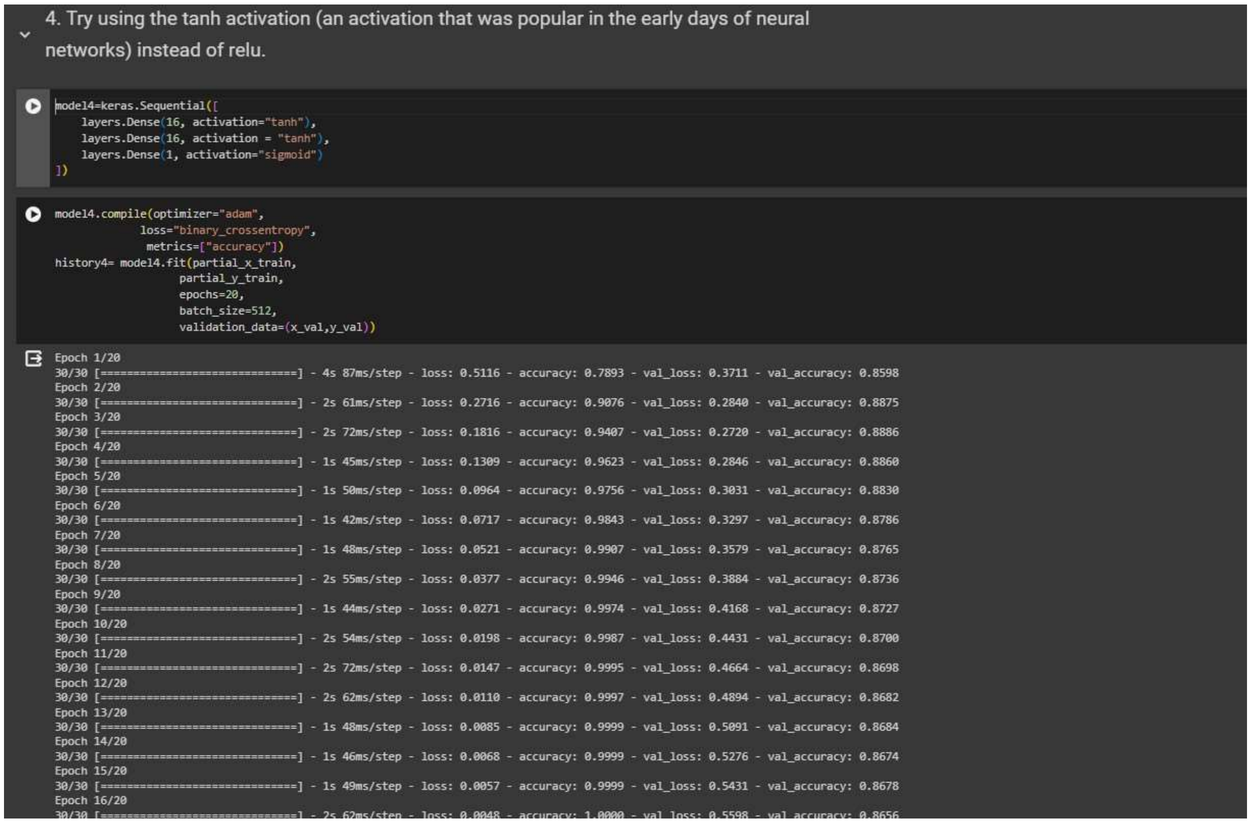
```
    Epoch 1/20
    30/30 [==============================] - 4s 87ms/step - loss: 2.0590e-04 - accuracy: 0.9999 - val_loss: 0.1160 - val_accuracy: 0.8659
    Epoch 2/20
    30/30 [==============================] - 2s 60ms/step - loss: 1.9826e-04 - accuracy: 0.9999 - val_loss: 0.1171 - val_accuracy: 0.8646
    Epoch 3/20
    30/30 [==============================] - 2s 58ms/step - loss: 1.5706e-04 - accuracy: 1.0000 - val_loss: 0.1179 - val_accuracy: 0.8638
    Epoch 4/20
    30/30 [==============================] - 2s 58ms/step - loss: 8.0067e-05 - accuracy: 1.0000 - val_loss: 0.1191 - val_accuracy: 0.8641
    Epoch 5/20
    30/30 [==============================] - 3s 84ms/step - loss: 4.3442e-05 - accuracy: 1.0000 - val_loss: 0.1197 - val_accuracy: 0.8640
    Epoch 6/20
    30/30 [==============================] - 1s 50ms/step - loss: 2.2530e-05 - accuracy: 1.0000 - val_loss: 0.1202 - val_accuracy: 0.8632
    Epoch 7/20
    30/30 [==============================] - 1s 49ms/step - loss: 1.4919e-05 - accuracy: 1.0000 - val_loss: 0.1204 - val_accuracy: 0.8621
    Epoch 8/20
    30/30 [==============================] - 2s 58ms/step - loss: 1.2111e-05 - accuracy: 1.0000 - val_loss: 0.1205 - val_accuracy: 0.8618
    Epoch 9/20
    30/30 [==============================] - 2s 60ms/step - loss: 1.0152e-05 - accuracy: 1.0000 - val_loss: 0.1207 - val_accuracy: 0.8623
    Epoch 10/20
    30/30 [==============================] - 2s 59ms/step - loss: 8.8359e-06 - accuracy: 1.0000 - val_loss: 0.1207 - val_accuracy: 0.8621
    Epoch 11/20
    30/30 [==============================] - 2s 52ms/step - loss: 7.8295e-06 - accuracy: 1.0000 - val_loss: 0.1208 - val_accuracy: 0.8622
    Epoch 12/20
    30/30 [==============================] - 2s 70ms/step - loss: 7.0037e-06 - accuracy: 1.0000 - val_loss: 0.1209 - val_accuracy: 0.8618
    Epoch 13/20
    30/30 [==============================] - 2s 67ms/step - loss: 6.3278e-06 - accuracy: 1.0000 - val_loss: 0.1210 - val_accuracy: 0.8621
    Epoch 14/20
    30/30 [==============================] - 2s 61ms/step - loss: 5.7782e-06 - accuracy: 1.0000 - val_loss: 0.1210 - val_accuracy: 0.8619
    Epoch 15/20
    30/30 [==============================] - 2s 60ms/step - loss: 5.2705e-06 - accuracy: 1.0000 - val_loss: 0.1211 - val_accuracy: 0.8621
    Epoch 16/20
    30/30 [==============================] - 2s 62ms/step - loss: 4.8376e-06 - accuracy: 1.0000 - val_loss: 0.1212 - val_accuracy: 0.8623
    Epoch 17/20
    30/30 [==============================] - 2s 59ms/step - loss: 4.4694e-06 - accuracy: 1.0000 - val_loss: 0.1213 - val_accuracy: 0.8623
    Epoch 18/20
    30/30 [==============================] - 1s 43ms/step - loss: 4.1370e-06 - accuracy: 1.0000 - val_loss: 0.1213 - val_accuracy: 0.8626
    Epoch 19/20
    30/30 [==============================] - 2s 64ms/step - loss: 3.8381e-06 - accuracy: 1.0000 - val_loss: 0.1214 - val_accuracy: 0.8623
    Epoch 20/20
    30/30 [==============================] - 2s 74ms/step - loss: 3.5826e-06 - accuracy: 1.0000 - val_loss: 0.1214 - val_accuracy: 0.8621
```

Instead of binary cross-entropy, the model uses Mean Squared Error (MSE) as its loss function. Additionally, the accuracy metric is kept for evaluation. The model is then trained using the same training and validation data for 20 epochs, with a batch size of 512. When comparing the performance of this model (let's call it model4) to the first model, several differences can be seen. First, the loss function is converted from binary cross-entropy to MSE. This change in loss function may cause differences in how the model updates its weights during training, particularly in how it handles misclassifications. As a result, model 4 has a significantly lower loss value than the first model, with a final loss of around 3.5826e-06.

By comparison, the validation accuracy for the final model is about 86.21%, which is a tiny reduction from the initial model. The alteration implies that although the model performs exceptionally well in fitting the training data (as demonstrated by the remarkably low loss and 100% training accuracy), it could not adapt to new data as effectively. With a marginally worse validation accuracy than the initial model, the variation in the loss function seems to have hindered the model's ability to generalize. While model 4's performance on the validation set suggests that it may be overfitting, overall, it achieves an astonishingly low loss on Training data.

4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.

Code:

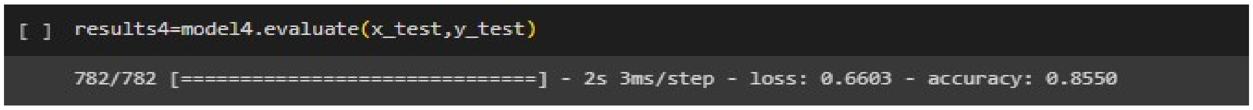


4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu.

```
model4=keras.Sequential([
    layers.Dense(16, activation="tanh"),
    layers.Dense(16, activation = "tanh"),
    layers.Dense(1, activation="sigmoid")
])
```

```
model4.compile(optimizer="adam",
              loss="binary_crossentropy",
               metrics=["accuracy"])
history4= model4.fit(partial_x_train,
                   partial_y_train,
                   epochs=20,
                   batch_size=512,
                   validation_data=(x_val,y_val))
```

```
Epoch 1/20
30/30 [==============================] - 4s 87ms/step - loss: 0.5116 - accuracy: 0.7893 - val_loss: 0.3711 - val_accuracy: 0.8598
Epoch 2/20
30/30 [==============================] - 2s 61ms/step - loss: 0.2716 - accuracy: 0.9076 - val_loss: 0.2840 - val_accuracy: 0.8875
Epoch 3/20
30/30 [==============================] - 2s 72ms/step - loss: 0.1816 - accuracy: 0.9407 - val_loss: 0.2720 - val_accuracy: 0.8886
Epoch 4/20
30/30 [==============================] - 1s 45ms/step - loss: 0.1309 - accuracy: 0.9623 - val_loss: 0.2846 - val_accuracy: 0.8860
Epoch 5/20
30/30 [==============================] - 1s 50ms/step - loss: 0.0964 - accuracy: 0.9756 - val_loss: 0.3031 - val_accuracy: 0.8830
Epoch 6/20
30/30 [==============================] - 1s 42ms/step - loss: 0.0717 - accuracy: 0.9843 - val_loss: 0.3297 - val_accuracy: 0.8786
Epoch 7/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0521 - accuracy: 0.9907 - val_loss: 0.3579 - val_accuracy: 0.8765
Epoch 8/20
30/30 [==============================] - 2s 55ms/step - loss: 0.0377 - accuracy: 0.9946 - val_loss: 0.3884 - val_accuracy: 0.8736
Epoch 9/20
30/30 [==============================] - 1s 44ms/step - loss: 0.0271 - accuracy: 0.9974 - val_loss: 0.4168 - val_accuracy: 0.8727
Epoch 10/20
30/30 [==============================] - 2s 54ms/step - loss: 0.0198 - accuracy: 0.9987 - val_loss: 0.4431 - val_accuracy: 0.8700
Epoch 11/20
30/30 [==============================] - 2s 72ms/step - loss: 0.0147 - accuracy: 0.9995 - val_loss: 0.4664 - val_accuracy: 0.8698
Epoch 12/20
30/30 [==============================] - 2s 62ms/step - loss: 0.0110 - accuracy: 0.9997 - val_loss: 0.4894 - val_accuracy: 0.8682
Epoch 13/20
30/30 [==============================] - 1s 48ms/step - loss: 0.0085 - accuracy: 0.9999 - val_loss: 0.5091 - val_accuracy: 0.8684
Epoch 14/20
30/30 [==============================] - 1s 46ms/step - loss: 0.0068 - accuracy: 0.9999 - val_loss: 0.5276 - val_accuracy: 0.8674
Epoch 15/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0057 - accuracy: 0.9999 - val_loss: 0.5431 - val_accuracy: 0.8678
Epoch 16/20
30/30 [==============================] - 2s 62ms/step - loss: 0.0048 - accuracy: 1.0000 - val_loss: 0.5598 - val_accuracy: 0.8656
```



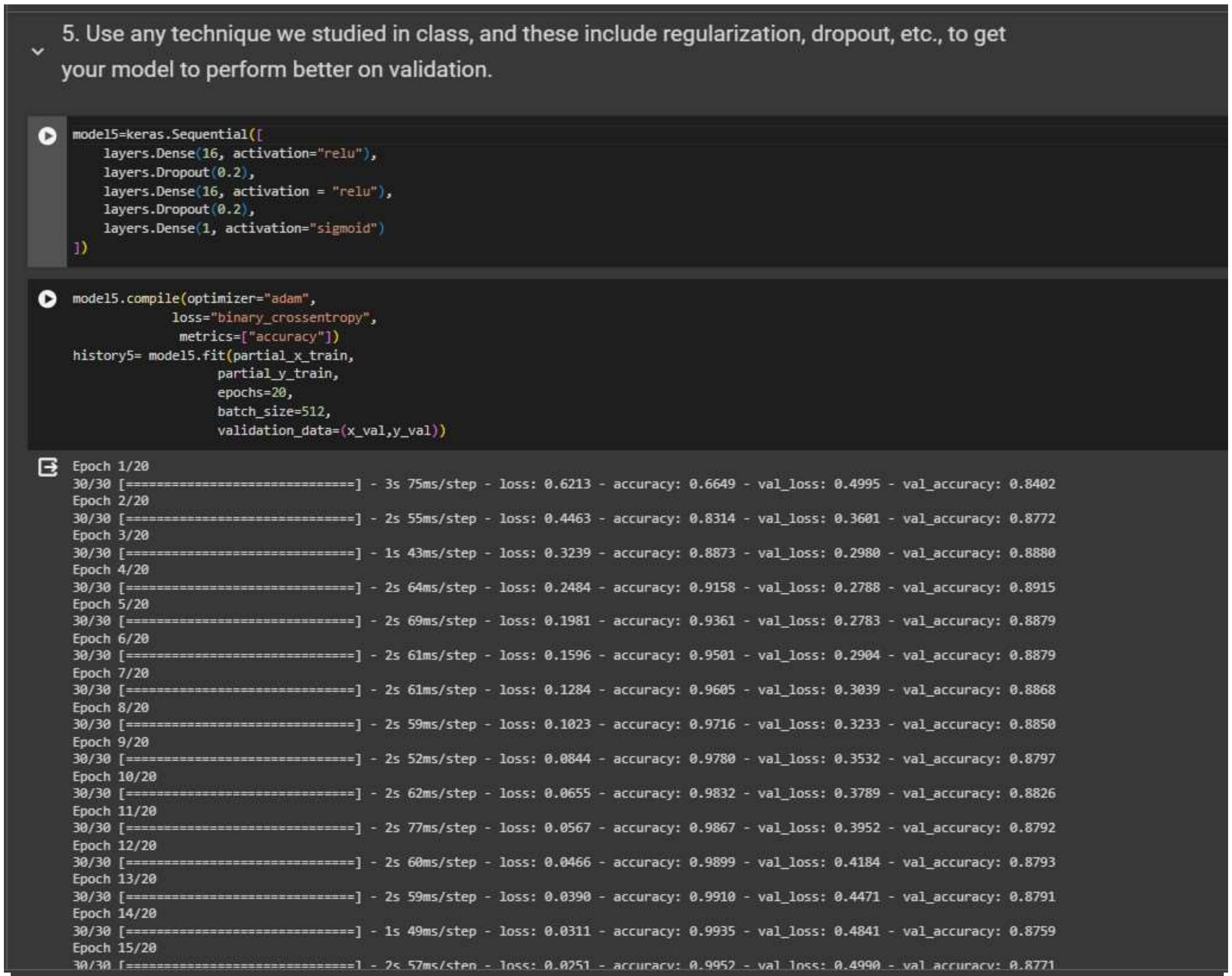
```
[ ]  results4=model4.evaluate(x_test,y_test)

     782/782 [==============================] - 2s 3ms/step - loss: 0.6603 - accuracy: 0.8550
```

Model4 uses tanh instead of ReLU for both hidden layers. The architecture remains the same, with two hidden layers of 16 units each and tanh activation functions, followed by a final output layer with sigmoid activation function for binary classification. The model is built using the Adam optimizer, a binary cross-entropy loss function, and accuracy as the metric. Similarly, it is trained using the same training and validation data for 20 epochs with a batch size of 512.

There are several differences between the performance of model 4 and the first model, which used ReLU activations. Model4 achieves a relatively high training accuracy of 100%, as does the first model, but its validation accuracy is slightly lower, with a final value of approximately 86.64%. Furthermore, model4's test accuracy is slightly lower than the first model, with a final accuracy of around 85.50%. This suggests that the choice of activation function influenced the model's ability to generalize to previously unseen data, with the tanh activation function performing slightly worse in terms of validation and test accuracy than ReLU. Overall, while both models achieve similar training accuracies, the first model with ReLU activations has slightly better generalization performance on the validation and test sets than model 4 with tanh activations.

5. Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.

Code:

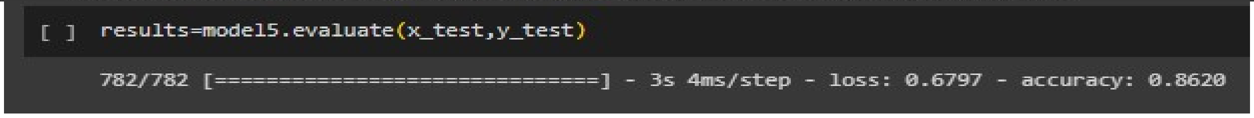


5. Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.

```
model5=keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dropout(0.2),
    layers.Dense(16, activation = "relu"),
    layers.Dropout(0.2),
    layers.Dense(1, activation="sigmoid")
])
```

```
model5.compile(optimizer="adam",
             loss="binary_crossentropy",
              metrics=["accuracy"])
history5= model5.fit(partial_x_train,
                   partial_y_train,
                   epochs=20,
                   batch_size=512,
                   validation_data=(x_val,y_val))
```

```
Epoch 1/20
30/30 [==============================] - 3s 75ms/step - loss: 0.6213 - accuracy: 0.6649 - val_loss: 0.4995 - val_accuracy: 0.8402
Epoch 2/20
30/30 [==============================] - 2s 55ms/step - loss: 0.4463 - accuracy: 0.8314 - val_loss: 0.3601 - val_accuracy: 0.8772
Epoch 3/20
30/30 [==============================] - 1s 43ms/step - loss: 0.3239 - accuracy: 0.8873 - val_loss: 0.2980 - val_accuracy: 0.8880
Epoch 4/20
30/30 [==============================] - 2s 64ms/step - loss: 0.2484 - accuracy: 0.9158 - val_loss: 0.2788 - val_accuracy: 0.8915
Epoch 5/20
30/30 [==============================] - 2s 69ms/step - loss: 0.1981 - accuracy: 0.9361 - val_loss: 0.2783 - val_accuracy: 0.8879
Epoch 6/20
30/30 [==============================] - 2s 61ms/step - loss: 0.1596 - accuracy: 0.9501 - val_loss: 0.2904 - val_accuracy: 0.8879
Epoch 7/20
30/30 [==============================] - 2s 61ms/step - loss: 0.1284 - accuracy: 0.9605 - val_loss: 0.3039 - val_accuracy: 0.8868
Epoch 8/20
30/30 [==============================] - 2s 59ms/step - loss: 0.1023 - accuracy: 0.9716 - val_loss: 0.3233 - val_accuracy: 0.8850
Epoch 9/20
30/30 [==============================] - 2s 52ms/step - loss: 0.0844 - accuracy: 0.9780 - val_loss: 0.3532 - val_accuracy: 0.8797
Epoch 10/20
30/30 [==============================] - 2s 62ms/step - loss: 0.0655 - accuracy: 0.9832 - val_loss: 0.3789 - val_accuracy: 0.8826
Epoch 11/20
30/30 [==============================] - 2s 77ms/step - loss: 0.0567 - accuracy: 0.9867 - val_loss: 0.3952 - val_accuracy: 0.8792
Epoch 12/20
30/30 [==============================] - 2s 60ms/step - loss: 0.0466 - accuracy: 0.9899 - val_loss: 0.4184 - val_accuracy: 0.8793
Epoch 13/20
30/30 [==============================] - 2s 59ms/step - loss: 0.0390 - accuracy: 0.9910 - val_loss: 0.4471 - val_accuracy: 0.8791
Epoch 14/20
30/30 [==============================] - 1s 49ms/step - loss: 0.0311 - accuracy: 0.9935 - val_loss: 0.4841 - val_accuracy: 0.8759
Epoch 15/20
30/30 [==============================] - 2s 57ms/step - loss: 0.0251 - accuracy: 0.9952 - val_loss: 0.4990 - val_accuracy: 0.8771
```



```
[ ]  results=model5.evaluate(x_test,y_test)

     782/782 [==============================] - 3s 4ms/step - loss: 0.6797 - accuracy: 0.8620
```

Model 5 is identified by the addition of dropout layers after each thick layer. Dropout is a regularization approach used in neural networks to minimize overfitting. It involves randomly discarding (setting to zero) a proportion of input units during training. To avoid overfitting, a 0.2-rate dropout happens after each hidden layer.

Model5's architecture is identical to the original model, with two 16-unit hidden layers and ReLU activation functions, followed by a final output layer with a sigmoid activation function for binary classification. The Adam optimizer is used to create the model, along with a binary cross-entropy loss function and accuracy as the measure.

Similarly, it is trained across 20 epochs using the same training and validation data with a batch size of 512. There are numerous variations in model5's performance compared to the first model. Model5 with dropout layers outperforms the initial model in generalization, with a final validation accuracy of 87.40% vs 86.21%

for the initial model. Both models achieve excellent training accuracy. Furthermore, model5 obtains a little higher test accuracy of roughly 86.20% compared to the initial model's test accuracy of about 85.58%. The introduction of dropout layers improves generalization performance by minimizing reliance on individual neurons and pushing the network to acquire stronger characteristics.

-Raghuveer Manusanipalli
811298559