Assignment 2: Neural Network

1. Summary

I developed thirteen models, including a base model, to explore how varying the model configurations would affect the performance metrics. These metrics, namely loss values and accuracy, were closely comparable across all thirteen models. The loss values ranged from 0.28 to 0.31, while accuracy scores varied between 86% and 88%.

First, I observed that increasing the number of hidden layers contributed to greater model complexity. This resulted in fewer training iterations (epochs) being required before the models began overfitting. Similarly, increasing the number of nodes in the hidden layers also reduced the number of epochs needed before overfitting occurred. Both of these factors—more layers and more nodes—essentially made the models more efficient in terms of convergence, but also more prone to overfitting.

When switching the loss function from 'binary_crossentropy' to 'mse', I noticed a slight drop in performance. I advise against using 'mse' for classification problems because its non-convex nature in binary classification makes it less suited for this task.

Forth, I experimented with using the 'tanh' activation function instead of 'ReLU'. While the models with 'tanh' reached the overfitting stage more quickly, those with 'ReLU' performed slightly better in terms of both loss values and accuracy. In the final step of model refinement, I implemented regularization and dropout techniques to improve generalization. I applied L2 regularization with a value of 0.000002, which led to better performance compared to the base model. Furthermore, a two-layer dropout method with a 50% dropout rate significantly reduced model complexity. This configuration resulted in the lowest loss value among all thirteen models, demonstrating the effectiveness of these techniques in reducing overfitting while maintaining strong performance.

	Model	Number of Hidden layers	Number of Nodes in Hidden layers	Activation function	Loss function	Epochs	Loss value	Accuracy	Technique
0	Base Model	2	16	Relu	binary_crossentropy	5	0.295277	0.88036	
1	Modified Model 1	1	16	Relu	binary_crossentropy	5	0.296580	0.88044	
2	Modified Model 2	3	16	Relu	binary_crossentropy	3	0.307072	0.88064	
3	Modified Model 3	2	8	Relu	binary_crossentropy	6	0.291259	0.88372	
4	Modified Model 4	2	32	Relu	binary_crossentropy	4	0.289604	0.88320	
5	Modified Model 5	2	64	Relu	binary_crossentropy	3	0.291381	0.88196	
6	Modified Model 6	2	16	Relu	mse	2	0.331524	0.86472	
7	Modified Model 7	2	8	tanh	binary_crossentropy	6	0.293252	0.87936	
8	Modified Model 8	2	16	tanh	binary_crossentropy	4	0.296740	0.87888	
9	Modified Model 9	2	32	tanh	binary_crossentropy	2	0.293387	0.87908	
10	Modified Model 10	2	64	tanh	binary_crossentropy	2	0.302332	0.87504	
11	Modified Model 11	2	16	Relu	binary_crossentropy	4	0.288519	0.88424	Regularization L2 at 0.000001
12	Modified Model 12	2	16	Relu	binary_crossentropy	7	0.288005	0.88248	Dropout at 50%

2. Models

2.1 Packages

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import regularizers
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
%matplotlib inline
```

[298] ✓ 0.0s Python

2.2 IMBD Data

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

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

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

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

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

[165] ✓ 4.4s Python

2.3 Based Model

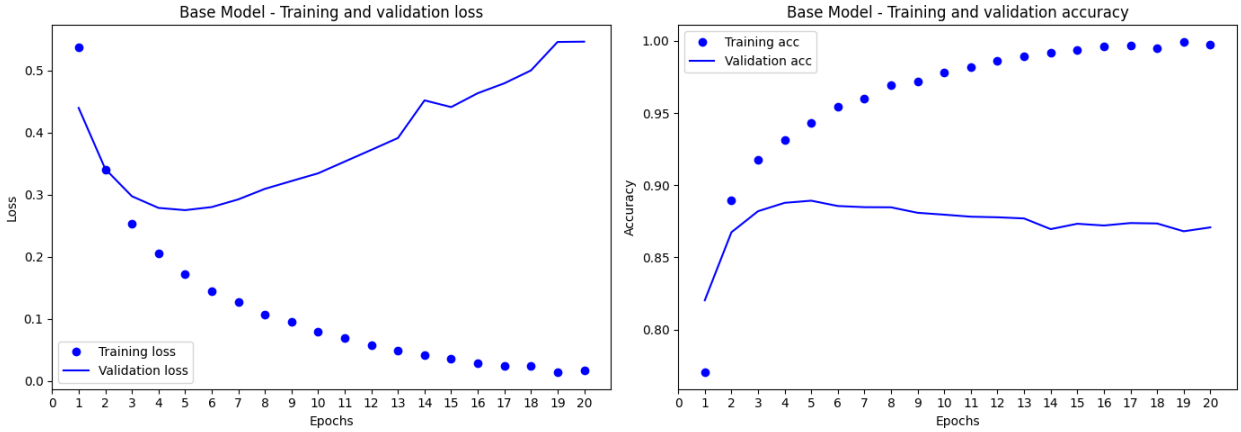
This model has 3 layers, 16 nodes for the first two layers with ‘Relu’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

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

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

based_model_history = based_model.fit(partial_x_train,
                                      partial_y_train,
                                      epochs=20,
                                      batch_size=512,
                                      validation_data=(x_val, y_val))
```

[166] ✓ 16.1s Python

```
# Plot
based_model_history_dict = based_model_history.history
based_model_loss_values = based_model_history_dict["loss"]
based_model_val_loss_values = based_model_history_dict["val_loss"]
based_model_epochs = range(1, len(based_model_loss_values) + 1)
modified_model_1_acc = based_model_history_dict["accuracy"]
modified_model_1_val_acc = based_model_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(based_model_epochs, based_model_loss_values, "bo", label="Training loss")
ax1.plot(based_model_epochs, based_model_val_loss_values, "b", label="Validation loss")
ax1.set_title("Base Model - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(based_model_epochs, modified_model_1_acc, "bo", label="Training acc")
ax2.plot(based_model_epochs, modified_model_1_val_acc, "b", label="Validation acc")
ax2.set_title("Base Model - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[175] ✓ 0.2s Python



The optimal number of epochs for **Base Model** is equal to 5. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 5.

The **Retrained Base Model** has the test accuracy equals to 88.04% and the test loss value of 0.2953.

```
based_model_R1 = keras.Sequential([
   layers.Dense(16, activation='relu'),
   layers.Dense(16, activation='relu'),
   layers.Dense(1, activation='sigmoid')
])

based_model_R1.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

based_model_R1_epoch = 5

based_model_R1.fit(partial_x_train,
                   partial_y_train,
                   epochs=based_model_R1_epoch,
                   batch_size=512,
                   validation_data=(x_val, y_val))
```

[342] ✓ 8.8s Python

```
Retrain_model_R1_results = based_model_R1.evaluate(x_test, y_test)
print("Retrained Base Model - Test Accuracy: ", round(Retrain_model_R1_results[1],4))
print("Retrained Base Model - Test Loss Value: ", round(Retrain_model_R1_results[0],4))
```

[343] ✓ 6.0s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8778 - loss: 0.2969
Retrained Base Model - Test Accuracy:  0.8804
Retrained Base Model - Test Loss Value:  0.2953
```

3 High vs Low number of hidden layers

A higher number of layers increases the complexity of the model, requiring fewer iterations (epochs) before entering the overfitting stage. I created two models: one with a single hidden layer of 16 nodes, and another with three hidden layers of 16 nodes each. I found that both models performed almost identically to the base model. The key difference is that the three-hidden-layer model reached the overfitting stage earlier, at epoch three, compared to the other model.

3.1 Modified Model 1

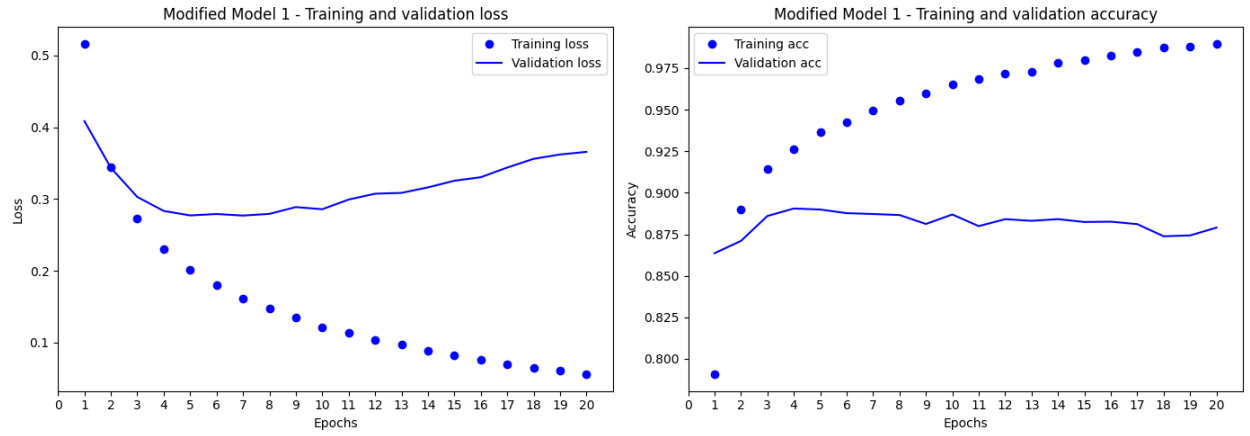
This model has 2 hidden layers with 16 nodes with ‘Relu’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

```
modified_model_1 = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_1.compile(optimizer='rmsprop',
                         loss = 'binary_crossentropy',
                         metrics=['accuracy'])

modified_model_1_history = modified_model_1.fit(partial_x_train,
                                                partial_y_train,
                                                epochs=20,
                                                batch_size=512,
                                                validation_data=(x_val, y_val))
```

[249] ✓ 13.5s Python

```
# Plot
modified_model_1_history_dict = modified_model_1_history.history
modified_model_1_loss_values = modified_model_1_history_dict["loss"]
modified_model_1_val_loss_values = modified_model_1_history_dict["val_loss"]
modified_model_1_epochs = range(1, len(modified_model_1_loss_values) + 1)
modified_model_1_acc = modified_model_1_history_dict["accuracy"]
modified_model_1_val_acc = modified_model_1_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_1_epochs, modified_model_1_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_1_epochs, modified_model_1_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 1 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_1_epochs, modified_model_1_acc, "bo", label="Training acc")
ax2.plot(modified_model_1_epochs, modified_model_1_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 1 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[177] ✓ 0.2s Python



The optimal number of epochs for **Modified Model 1** is equal to 5. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 5.

The **Retrained Modeified Model 1** has the test accuracy equals to 88.04% and the test loss value of 0.2966.

```
modified_model_R1 = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R1.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_R1_epoch = 5

modified_model_R1.fit(partial_x_train,
                                partial_y_train,
                                epochs=modified_model_R1_epoch,
                                batch_size=512,
                                validation_data=(x_val, y_val))
```

[251] ✓ 6.5s Python

```
modified_model_R1_results = modified_model_R1.evaluate(x_test, y_test)
print("Retrained Modeified Model 1 - Test Accuracy: ", round(modified_model_R1_results[1],4))
print("Retrained Modifired Model 1 - Test Loss Value: ", round(modified_model_R1_results[0],4))
```

[252] ✓ 5.8s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8791 - loss: 0.2964
Retrained Modeified Model 1 - Test Accuracy:  0.8804
Retrained Modifired Model 1 - Test Loss Value:  0.2966
```

3.2 Modified Model 2

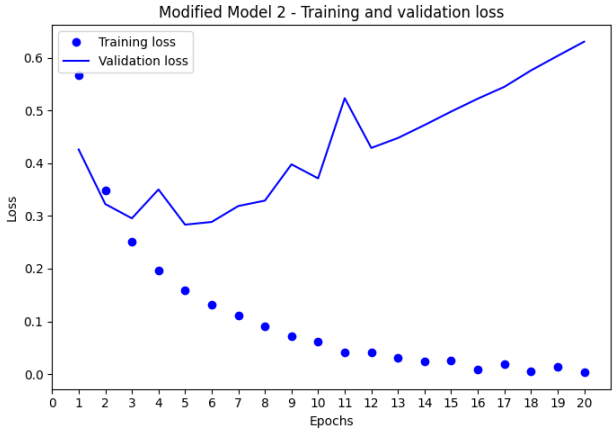
This model has 3 hidden layers of 16 nodes with ‘Relu’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

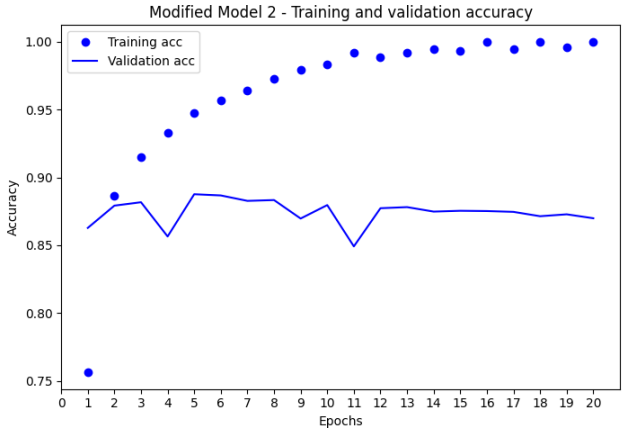
```
modified_model_2 = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_2.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_2_history = modified_model_2.fit(partial_x_train,
                                partial_y_train,
                                epochs=20,
                                batch_size=512,
                                validation_data=(x_val, y_val))
```

[255] ✓ 14.3s Python

```
# Plot
modified_model_2_history_dict = modified_model_2_history.history
modified_model_2_loss_values = modified_model_2_history_dict["loss"]
modified_model_2_val_loss_values = modified_model_2_history_dict["val_loss"]
modified_model_2_epochs = range(1, len(modified_model_2_loss_values) + 1)
modified_model_2_acc = modified_model_2_history_dict["accuracy"]
modified_model_2_val_acc = modified_model_2_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_2_epochs, modified_model_2_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_2_epochs, modified_model_2_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 2 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_2_epochs, modified_model_2_acc, "bo", label="Training acc")
ax2.plot(modified_model_2_epochs, modified_model_2_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 2 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[158] ✓ 0.1s Python





The optimal number of epochs for **Modified Model 2** is equal to 3. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 3.

The **Retrained Modeified Model 2** has the test accuracy equals to 88.06% and the test loss value of 0.3071.

```
modified_model_R2 = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R2.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_R2_epoch = 3

modified_model_R2.fit(partial_x_train,
                      partial_y_train,
                      epochs=modified_model_R2_epoch,
                      batch_size=512,
                      validation_data=(x_val, y_val))
```

[257] ✓ 5.9s Python

```
modified_model_R2_results = modified_model_R2.evaluate(x_test, y_test)
print("Retrained Modeified Model 2 - Test Accuracy: ", round(modified_model_R2_results[1],4))
print("Retrained Modifired Model 2 - Test Loss Value: ", round(modified_model_R2_results[0],4))
```

[258] ✓ 5.9s Python

```
··· 782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8813 - loss: 0.3084
Retrained Modeified Model 2 - Test Accuracy:  0.8806
Retrained Modifired Model 2 - Test Loss Value:  0.3071
```

4 More VS Less nodes in the hidden layers

Similar to the number of layers, increasing the number of nodes also increases the model's complexity, leading to earlier overfitting compared to models with fewer nodes. The base model has 16 nodes in each hidden layer, so I created three additional models with 8, 32, and 64 nodes to compare. As expected, the model with 64 nodes reached the overfitting stage at epoch 3, while the model with 8 nodes reached it at epoch 6. The performance metrics were almost identical across the models.

4.1 Modified Model 3

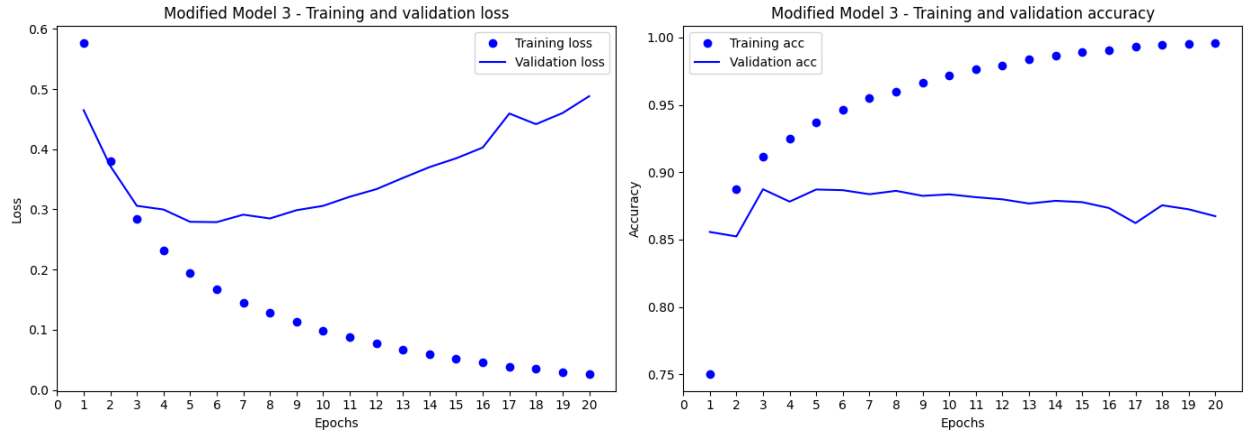
This model has 2 hidden layers, 8 nodes for the first two layers with ‘relu’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

```
modified_model_3 = keras.Sequential([
    layers.Dense(8, activation='relu'),
    layers.Dense(8, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_3.compile(optimizer='rmsprop',
                         loss = 'binary_crossentropy',
                         metrics=['accuracy'])

modified_model_3_history = modified_model_3.fit(partial_x_train,
                                                partial_y_train,
                                                epochs=20,
                                                batch_size=512,
                                                validation_data=(x_val, y_val))
```

[259] ✓ 15.5s Python

```
# Plot
modified_model_3_history_dict = modified_model_3_history.history
modified_model_3_loss_values = modified_model_3_history_dict["loss"]
modified_model_3_val_loss_values = modified_model_3_history_dict["val_loss"]
modified_model_3_epochs = range(1, len(modified_model_3_loss_values) + 1)
modified_model_3_acc = modified_model_3_history_dict["accuracy"]
modified_model_3_val_acc = modified_model_3_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_3_epochs, modified_model_3_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_3_epochs, modified_model_3_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 3 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_3_epochs, modified_model_3_acc, "bo", label="Training acc")
ax2.plot(modified_model_3_epochs, modified_model_3_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 3 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[183] ✓ 0.1s Python



The optimal number of epochs for **Modified Model 3** is equal to 6. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 6.

The **Retrained Modeified Model 3** has the test accuracy equals to 88.37% and the test loss value of 0.2913.

```
modified_model_R3 = keras.Sequential([
    layers.Dense(8, activation='relu'),
    layers.Dense(8, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R3.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_R3_epoch = 6

modified_model_R3.fit(partial_x_train,
                                partial_y_train,
                                epochs=modified_model_R3_epoch,
                                batch_size=512,
                                validation_data=(x_val, y_val))
```

[261] ✓ 7.9s Python

```
modified_model_R3_results = modified_model_R3.evaluate(x_test, y_test)
print("Retrained Modeified Model 3 - Test Accuracy: ", round(modified_model_R3_results[1],4))
print("Retrained Modifired Model 3 - Test Loss Value: ", round(modified_model_R3_results[0],4))
```

[262] ✓ 5.3s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8818 - loss: 0.2923
Retrained Modeified Model 3 - Test Accuracy:  0.8837
Retrained Modifired Model 3 - Test Loss Value:  0.2913
```

4.2 Modified Model 4

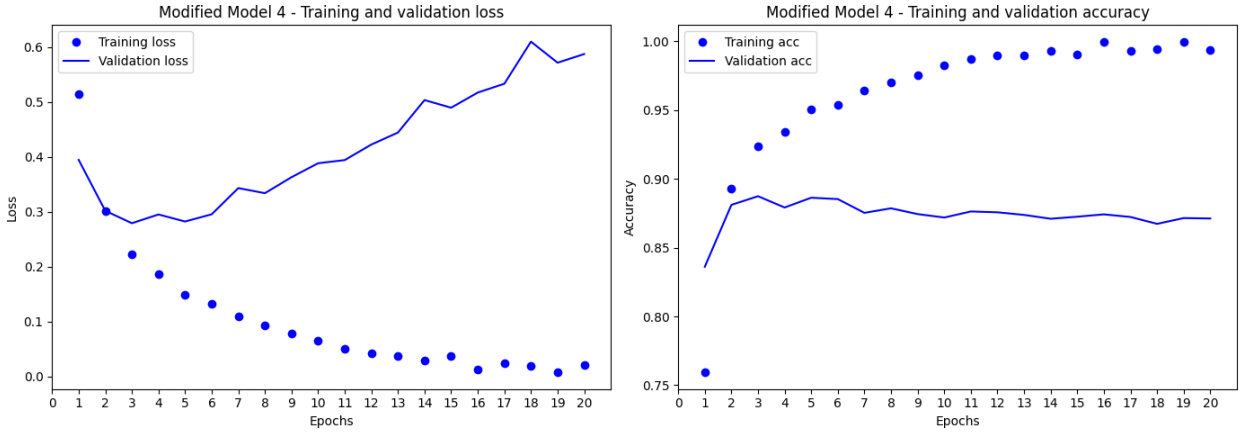
This model has 2 hidden layers of 32 nodes with ‘relu’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

```
modified_model_4 = keras.Sequential([
    layers.Dense(32, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_4.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_4_history = modified_model_4.fit(partial_x_train,
                                partial_y_train,
                                epochs=20,
                                batch_size=512,
                                validation_data=(x_val, y_val))
```

[263] ✓ 13.6s Python

```
# Plot
modified_model_4_history_dict = modified_model_4_history.history
modified_model_4_loss_values = modified_model_4_history_dict["loss"]
modified_model_4_val_loss_values = modified_model_4_history_dict["val_loss"]
modified_model_4_epochs = range(1, len(modified_model_4_loss_values) + 1)
modified_model_4_acc = modified_model_4_history_dict["accuracy"]
modified_model_4_val_acc = modified_model_4_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_4_epochs, modified_model_4_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_4_epochs, modified_model_4_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 4 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_4_epochs, modified_model_4_acc, "bo", label="Training acc")
ax2.plot(modified_model_4_epochs, modified_model_4_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 4 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[187] ✓ 0.1s Python



The optimal number of epochs for **Modified Model 4** is equal to 4. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 4.

The **Retrained Modeified Model 4** has the test accuracy equals to 88.32% and the test loss value of 0.2896.

```
modified_model_R4 = keras.Sequential([
    layers.Dense(32, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R4.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_R4_epoch = 4

modified_model_R4.fit(partial_x_train,
                      partial_y_train,
                      epochs=modified_model_R4_epoch,
                      batch_size=512,
                      validation_data=(x_val, y_val))
```

[267] ✓ 7.1s Python

```
modified_model_R4_results = modified_model_R4.evaluate(x_test, y_test)
print("Retrained Modeified Model 4 - Test Accuracy: ", round(modified_model_R4_results[1],4))
print("Retrained Modifired Model 4 - Test Loss Value: ", round(modified_model_R4_results[0],4))
```

[358] ✓ 6.6s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.8799 - loss: 0.2910
Retrained Modeified Model 4 - Test Accuracy:  0.8832
Retrained Modifired Model 4 - Test Loss Value:  0.2896
```

4.3 Modified Model 5

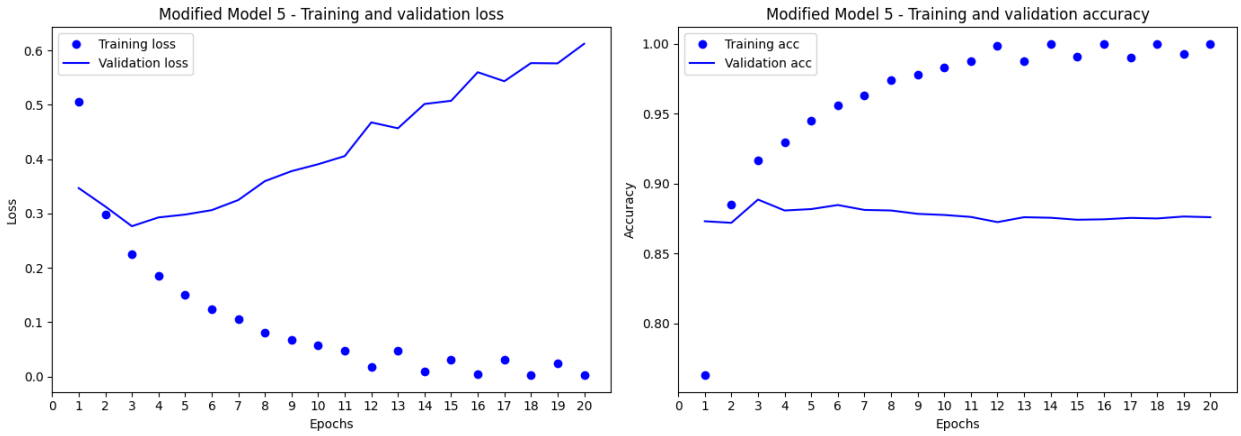
This model has 2 hidden layers of 64 nodes with ‘relu’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

```
modified_model_5 = keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_5.compile(optimizer='rmsprop',
                         loss = 'binary_crossentropy',
                         metrics=['accuracy'])

modified_model_5_history = modified_model_5.fit(partial_x_train,
                                                partial_y_train,
                                                epochs=20,
                                                batch_size=512,
                                                validation_data=(x_val, y_val))
```

[269] ✓ 13.5s Python

```python
# Plot
modified_model_5_history_dict = modified_model_5_history.history
modified_model_5_loss_values = modified_model_5_history_dict["loss"]
modified_model_5_val_loss_values = modified_model_5_history_dict["val_loss"]
modified_model_5_epochs = range(1, len(modified_model_5_loss_values) + 1)
modified_model_5_acc = modified_model_5_history_dict["accuracy"]
modified_model_5_val_acc = modified_model_5_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_5_epochs, modified_model_5_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_5_epochs, modified_model_5_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 5 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_5_epochs, modified_model_5_acc, "bo", label="Training acc")
ax2.plot(modified_model_5_epochs, modified_model_5_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 5 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[191] ✓ 0.1s Python



The optimal number of epochs for **Modified Model 5** is equal to 23. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 3.

The **Retrained Modeified Model 5** has the test accuracy equals to 88.2% and the test loss value of 0.2914.

```python
modified_model_R5 = keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R5.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_R5_epoch = 3

modified_model_R5.fit(partial_x_train,
                                  partial_y_train,
                                  epochs=modified_model_R5_epoch,
                                  batch_size=512,
                                  validation_data=(x_val, y_val))
```

[271] ✓ 6.0s Python

```
modified_model_R5_results = modified_model_R5.evaluate(x_test, y_test)
print("Retrained Modeified Model 5 - Test Accuracy: ", round(modified_model_R5_results[1],4))
print("Retrained Modifired Model 5 - Test Loss Value: ", round(modified_model_R5_results[0],4))
```

[272] ✓ 5.6s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8794 - loss: 0.2937
Retrained Modeified Model 5 - Test Accuracy:  0.882
Retrained Modifired Model 5 - Test Loss Value:  0.2914
```

5 Loss function; 'binary_crossentopy' VS 'mse'

This is a binary classification problem, and the appropriate loss function is 'binary_crossentropy'. While 'mse' can be used as a loss function for this type of problem, it is not recommended due to the non-convexity in binary classification. Although the model with 'mse' overfitted early at epoch 3, its performance was inferior to the other models, with an accuracy of 86.47% and a loss value of 0.3315.

5.1 Modified Model 6

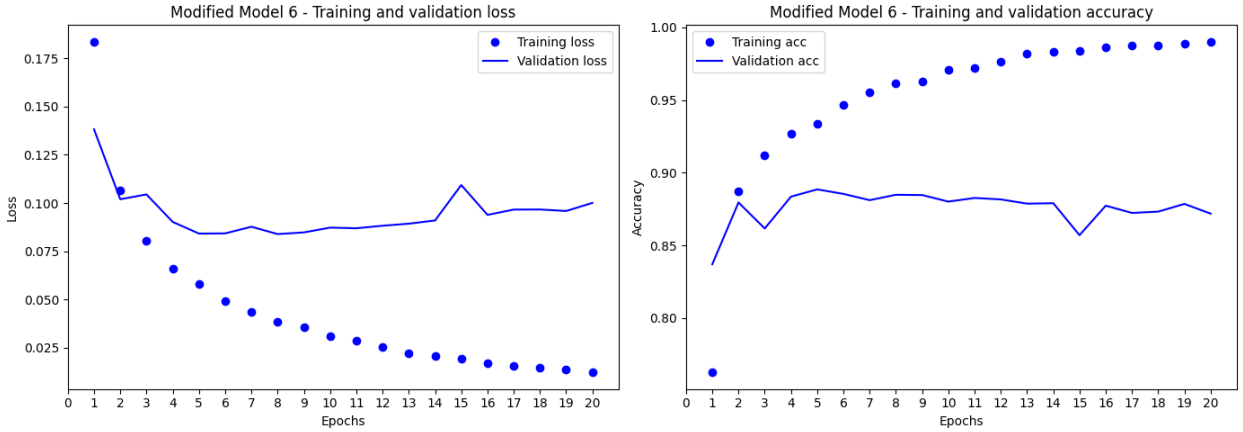
This model has 2 hidden layers of 16 nodes with 'Relu' activation function, and a single node for the last layer with 'Sigmoid' activation function. The loss function is 'mse' instead of 'binary_crossentropy'.

```
modified_model_6 = keras.Sequential([
   layers.Dense(16, activation='relu'),
   layers.Dense(16, activation='relu'),
   layers.Dense(1, activation='sigmoid')
])

modified_model_6.compile(optimizer='rmsprop',
                         loss = 'mse',
                         metrics=['accuracy'])

modified_model_6_history = modified_model_6.fit(partial_x_train,
                                                partial_y_train,
                                                epochs=20,
                                                batch_size=512,
                                                validation_data=(x_val, y_val))
```

[273] ✓ 13.1s Python

```
# Plot
modified_model_6_history_dict = modified_model_6_history.history
modified_model_6_loss_values = modified_model_6_history_dict["loss"]
modified_model_6_val_loss_values = modified_model_6_history_dict["val_loss"]
modified_model_6_epochs = range(1, len(modified_model_6_loss_values) + 1)
modified_model_6_acc = modified_model_6_history_dict["accuracy"]
modified_model_6_val_acc = modified_model_6_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_6_epochs, modified_model_6_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_6_epochs, modified_model_6_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 6 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_6_epochs, modified_model_6_acc, "bo", label="Training acc")
ax2.plot(modified_model_6_epochs, modified_model_6_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 6 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[199] ✓ 0.1s Python



The optimal number of epochs for **Modified Model 6** is equal to 2. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 2.

The **Retrained Modeified Model 6** has the test accuracy equals to 86.47% and the test loss value of 0.3315.

```
modified_model_R6 = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R6.compile(optimizer='rmsprop',
                          loss = 'mse',
                          metrics=['accuracy'])

modified_model_R6_epoch = 2

modified_model_R6_history = modified_model_R6.fit(partial_x_train,
                                                  partial_y_train,
                                                  epochs=modified_model_R6_epoch,
                                                  batch_size=512,
                                                  validation_data=(x_val, y_val))
```

[280] ✓ 4.9s Python

```
modified_model_R6_results = modified_model_R4.evaluate(x_test, y_test)
print("Retrained Modeified Model 6 - Test Accuracy: ", round(modified_model_R6_results[1],4))
print("Retrained Modifired Model 6 - Test Loss Value: ", round(modified_model_R6_results[0],4))
```

[282] ✓ 4.7s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8650 - loss: 0.3294
Retrained Modeified Model 6 - Test Accuracy:  0.8647
Retrained Modifired Model 6 - Test Loss Value:  0.3315
```

6 Activation function; ReLU vs Tanh

Both ReLU and Tanh are very popular activation functions, each with its own unique use case. Tanh is particularly effective when dealing with negative values or in shallow networks. I created four models, varying the number of nodes from 8 to 64. I found that for all the models, whether using ‘ReLU’ or ‘Tanh’, the performance was nearly identical. The only difference was that ‘Tanh’ tended to reach overfitting slightly more easily.

6.1 Modified Model 7

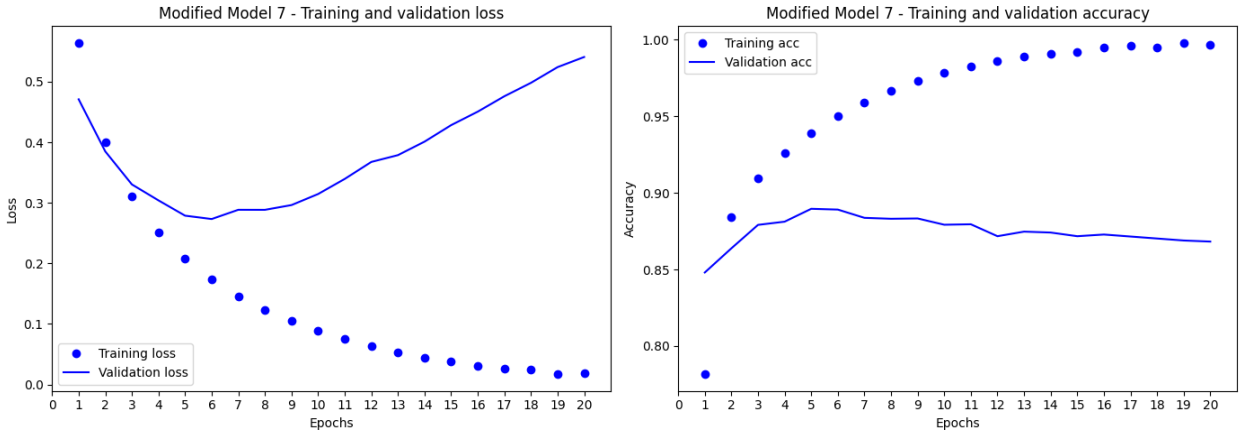
This model has 2 hidden layers of 8 nodes with ‘tanh’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

```python
modified_model_7 = keras.Sequential([
    layers.Dense(8, activation='tanh'),
    layers.Dense(8, activation='tanh'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_7.compile(optimizer='rmsprop',
                         loss = 'binary_crossentropy',
                         metrics=['accuracy'])

modified_model_7_history = modified_model_7.fit(partial_x_train,
                                                partial_y_train,
                                                epochs=20,
                                                batch_size=512,
                                                validation_data=(x_val, y_val))
```

[277] ✓ 16.3s Python

```python
# Plot
modified_model_7_history_dict = modified_model_7_history.history
modified_model_7_loss_values = modified_model_7_history_dict["loss"]
modified_model_7_val_loss_values = modified_model_7_history_dict["val_loss"]
modified_model_7_epochs = range(1, len(modified_model_7_loss_values) + 1)
modified_model_7_acc = modified_model_7_history_dict["accuracy"]
modified_model_7_val_acc = modified_model_7_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_7_epochs, modified_model_7_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_7_epochs, modified_model_7_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 7 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_7_epochs, modified_model_7_acc, "bo", label="Training acc")
ax2.plot(modified_model_7_epochs, modified_model_7_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 7 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[203] ✓ 0.2s Python



The optimal number of epochs for **Modified Model 7** is equal to 6. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 6.

The **Retrained Modified Model 7** has the test accuracy equals to 87.94% and the test loss value of 0.2933.

```
modified_model_R1 = keras.Sequential([
   layers.Dense(16, activation='relu'),
   layers.Dense(1, activation='sigmoid')
])

modified_model_R1.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

modified_model_R1_epoch = 5

modified_model_R1.fit(partial_x_train,
                      partial_y_train,
                      epochs=modified_model_R1_epoch,
                      batch_size=512,
                      validation_data=(x_val, y_val))
```

[251] ✓ 6.5s Python

```
modified_model_R7_results = modified_model_R7.evaluate(x_test, y_test)
print("Retrained Modeified Model 7 - Test Accuracy: ", round(modified_model_R7_results[1],4))
print("Retrained Modifired Model 7 - Test Loss Value: ", round(modified_model_R7_results[0],4))
```

[285] ✓ 5.5s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8779 - loss: 0.2949
Retrained Modeified Model 7 - Test Accuracy:  0.8794
Retrained Modifired Model 7 - Test Loss Value:  0.2933
```

6.2 Modified Model 8

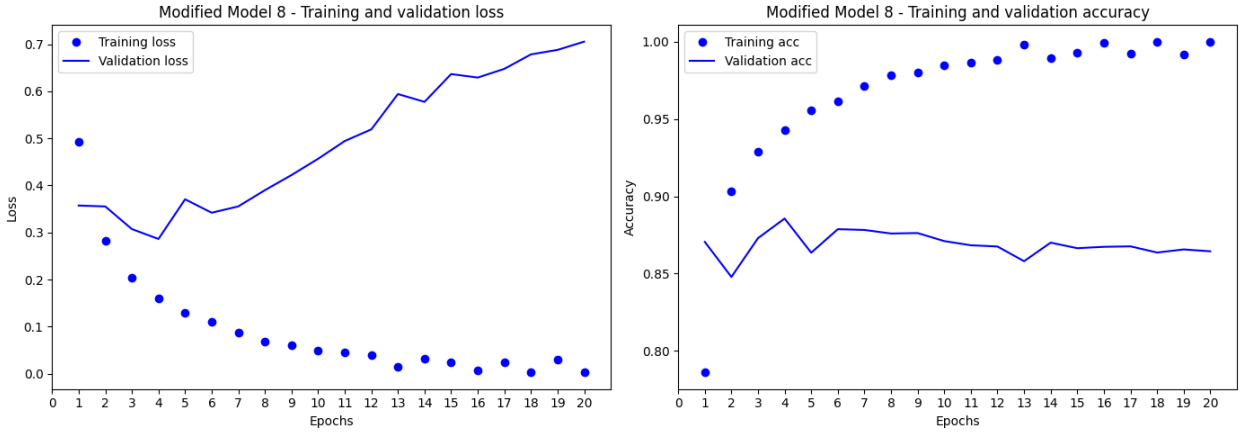
This model has 2 hidden layers of 16 nodes with ‘tanh’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

```
modified_model_8 = keras.Sequential([
   layers.Dense(16, activation='tanh'),
   layers.Dense(16,  activation='tanh'),
   layers.Dense(1, activation='sigmoid')
])

modified_model_8.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

modified_model_8_history = modified_model_8.fit(partial_x_train,
                      partial_y_train,
                      epochs=20,
                      batch_size=512,
                      validation_data=(x_val, y_val))
```

[286] ✓ 14.2s Python

```
# Plot
modified_model_8_history_dict = modified_model_8_history.history
modified_model_8_loss_values = modified_model_8_history_dict["loss"]
modified_model_8_val_loss_values = modified_model_8_history_dict["val_loss"]
modified_model_8_epochs = range(1, len(modified_model_8_loss_values) + 1)
modified_model_8_acc = modified_model_8_history_dict["accuracy"]
modified_model_8_val_acc = modified_model_8_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_8_epochs, modified_model_8_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_8_epochs, modified_model_8_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 8 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_8_epochs, modified_model_8_acc, "bo", label="Training acc")
ax2.plot(modified_model_8_epochs, modified_model_8_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 8 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[287] ✓ 0.1s Python



The optimal number of epochs for **Modified Model 8** is equal to 4. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 4.

The **Retrained Modified Model 8** has the test accuracy equals to 87.89% and the test loss value of 0.2967.

```
modified_model_R8 = keras.Sequential([
   layers.Dense(16, activation='tanh'),
   layers.Dense(16, activation='tanh'),
   layers.Dense(1, activation='sigmoid')
])

modified_model_R8.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

modified_model_R8_epoch = 4

modified_model_R8_history = modified_model_R8.fit(partial_x_train,
                        partial_y_train,
                        epochs=modified_model_R8_epoch,
                        batch_size=512,
                        validation_data=(x_val, y_val))
```

[288] ✓ 6.5s Python

```
modified_model_R8_results = modified_model_R8.evaluate(x_test, y_test)
print("Retrained Modeified Model 8 - Test Accuracy: ", round(modified_model_R8_results[1],4))
print("Retrained Modifired Model 8 - Test Loss Value: ", round(modified_model_R8_results[0],4))
```

[289] ✓ 5.6s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8779 - loss: 0.2967
Retrained Modeified Model 8 - Test Accuracy:  0.8789
Retrained Modifired Model 8 - Test Loss Value:  0.2967
```

6.3 Modified Model 9

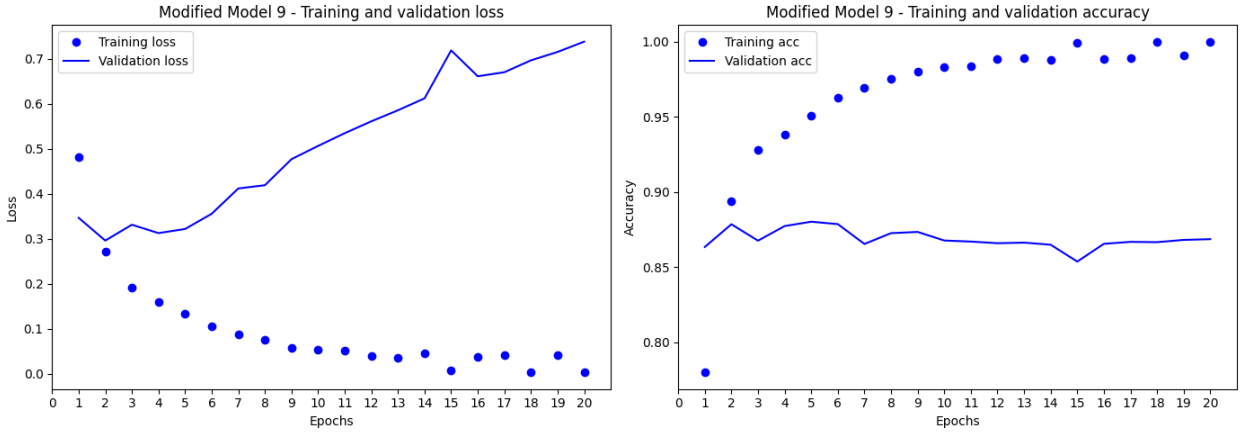
This model has 2 hidden layers of 32 nodes with ‘tanh’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

```
modified_model_9 = keras.Sequential([
   layers.Dense(32, activation='tanh'),
   layers.Dense(32,  activation='tanh'),
   layers.Dense(1, activation='sigmoid')
])

modified_model_9.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

modified_model_9_history = modified_model_9.fit(partial_x_train,
                        partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val, y_val))
```

[290] ✓ 13.6s Python

```python
# Plot
modified_model_9_history_dict = modified_model_9_history.history
modified_model_9_loss_values = modified_model_9_history_dict["loss"]
modified_model_9_val_loss_values = modified_model_9_history_dict["val_loss"]
modified_model_9_epochs = range(1, len(modified_model_9_loss_values) + 1)
modified_model_9_acc = modified_model_9_history_dict["accuracy"]
modified_model_9_val_acc = modified_model_9_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_9_epochs, modified_model_9_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_9_epochs, modified_model_9_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 9 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_9_epochs, modified_model_9_acc, "bo", label="Training acc")
ax2.plot(modified_model_9_epochs, modified_model_9_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 9 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[291] ✓ 0.1s Python



The optimal number of epochs for **Modified Model 9** is equal to 2. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 2.

The **Retrained Modified Model 9** has the test accuracy equals to 87.91% and the test loss value of 0.2934.

```
modified_model_R9 = keras.Sequential([
    layers.Dense(32, activation='tanh'),
    layers.Dense(32, activation='tanh'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R9.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_R9_epoch = 2

modified_model_R9_history = modified_model_R9.fit(partial_x_train,
                                                  partial_y_train,
                                                  epochs=modified_model_R9_epoch,
                                                  batch_size=512,
                                                  validation_data=(x_val, y_val))
```

[348] ✓ 7.2s Python

```
modified_model_R9_results = modified_model_R9.evaluate(x_test, y_test)
print("Retrained Modeified Model 9 - Test Accuracy: ", round(modified_model_R9_results[1],4))
print("Retrained Modifired Model 9 - Test Loss Value: ", round(modified_model_R9_results[0],4))
```

[349] ✓ 6.3s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.8798 - loss: 0.2925
Retrained Modeified Model 9 - Test Accuracy:  0.8791
Retrained Modifired Model 9 - Test Loss Value:  0.2934
```

6.4 Modified Model 10

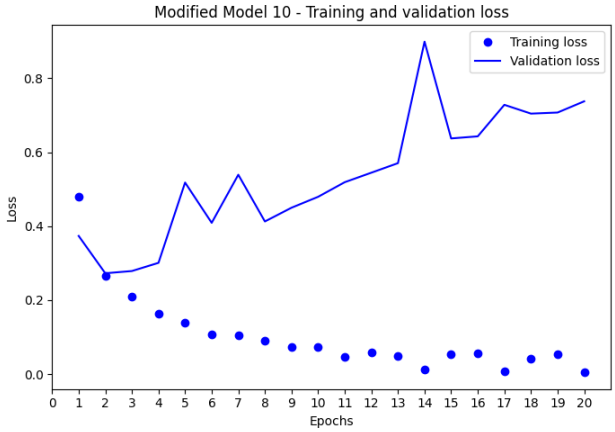
This model has 2 hidden layers of 64 nodes with ‘tanh’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function.

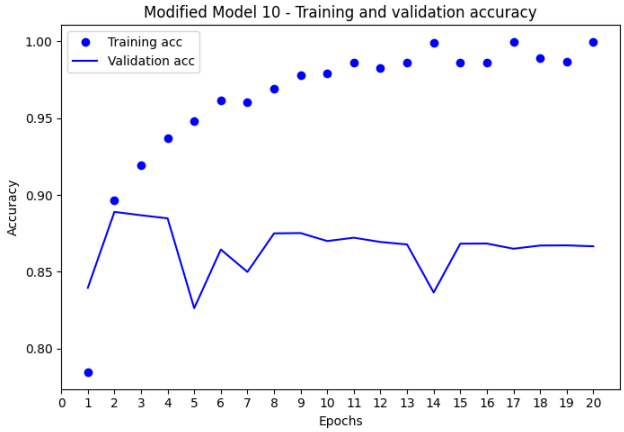
```
modified_model_10 = keras.Sequential([
    layers.Dense(64, activation='tanh'),
    layers.Dense(64,  activation='tanh'),
    layers.Dense(1, activation='sigmoid')
])

modified_model_10.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_10_history = modified_model_10.fit(partial_x_train,
                                                  partial_y_train,
                                                  epochs=20,
                                                  batch_size=512,
                                                  validation_data=(x_val, y_val))
```

[294] ✓ 13.5s Python

```
# Plot
modified_model_10_history_dict = modified_model_10_history.history
modified_model_10_loss_values = modified_model_10_history_dict["loss"]
modified_model_10_val_loss_values = modified_model_10_history_dict["val_loss"]
modified_model_10_epochs = range(1, len(modified_model_10_loss_values) + 1)
modified_model_10_acc = modified_model_10_history_dict["accuracy"]
modified_model_10_val_acc = modified_model_10_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_10_epochs, modified_model_10_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_10_epochs, modified_model_10_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 10 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_10_epochs, modified_model_10_acc, "bo", label="Training acc")
ax2.plot(modified_model_10_epochs, modified_model_10_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 10 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[295] ✓ 0.1s Python





The optimal number of epochs for **Modified Model 10** is equal to 2. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 2.

The **Retrained Modified Model 10** has the test accuracy equals to 87.5% and the test loss value of 0.3023.

```
modified_model_R10 = keras.Sequential([
   layers.Dense(64, activation='tanh'),
   layers.Dense(64, activation='tanh'),
   layers.Dense(1, activation='sigmoid')
])

modified_model_R10.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

modified_model_R10_epoch = 2

modified_model_R10_history = modified_model_R10.fit(partial_x_train,
                                    partial_y_train,
                                    epochs=modified_model_R10_epoch,
                                    batch_size=512,
                                    validation_data=(x_val, y_val))
```

[296] ✓ 5.7s Python

```
Epoch 1/2
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 81ms/step - accuracy: 0.6914 - loss: 0.5877 - val_accuracy: 0.8741 - val_loss: 0.3184
Epoch 2/2
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8933 - loss: 0.2759 - val_accuracy: 0.8808 - val_loss: 0.2886
```

```
modified_model_R10_results = modified_model_R10.evaluate(x_test, y_test)
print("Retrained Modeified Model 10 - Test Accuracy: ", round(modified_model_R10_results[1],4))
print("Retrained Modifired Model 10 - Test Loss Value: ", round(modified_model_R10_results[0],4))
```

[297] ✓ 5.4s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.8773 - loss: 0.2986
Retrained Modeified Model 10 - Test Accuracy:  0.875
Retrained Modifired Model 10 - Test Loss Value:  0.3023
```

7 Model generalization: Regularization and Dropout

We generally don't want the model to be too complex, so generalization techniques play a crucial role in reducing complexity. There are multiple methods available, but I created two models that incorporate regularization and dropout as generalization techniques. For this particular problem, the dropout method worked exceptionally well. Both models showed promising performance compared to the others. Specifically, with the dropout method, I observed a significant increase in the number of epochs required, indicating reduced complexity.

7.1 Regularization

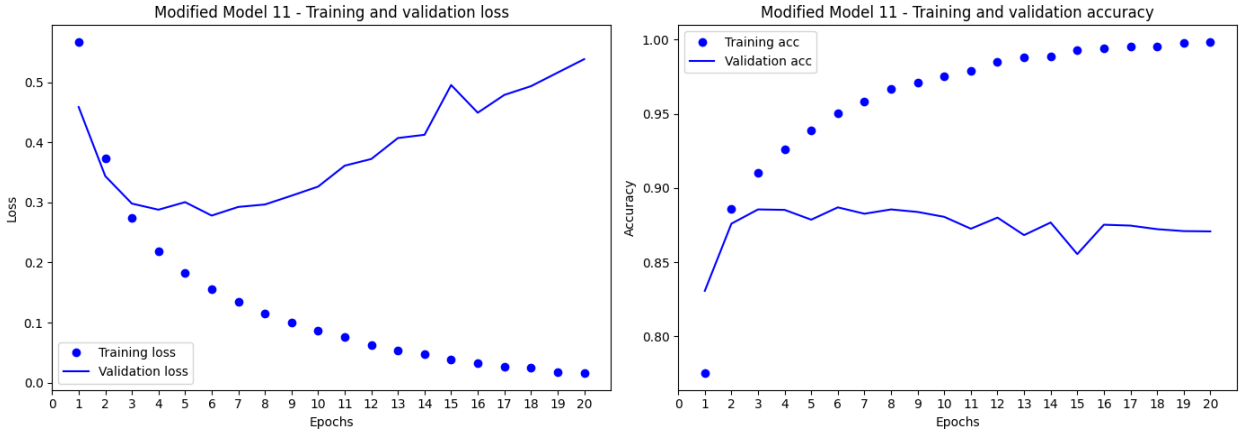
This model has 2 hidden layers of 16 nodes with ‘relu’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function. I did the L2 (squared form) regularization with the value of 0.000001.

```
modified_model_11 = keras.Sequential([
    layers.Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.000001)),
    layers.Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.000001)),
    layers.Dense(1, activation='sigmoid')
])

modified_model_11.compile(optimizer='rmsprop',
                          loss = 'binary_crossentropy',
                          metrics=['accuracy'])

modified_model_11_history = modified_model_11.fit(partial_x_train,
                                                  partial_y_train,
                                                  epochs=20,
                                                  batch_size=512,
                                                  validation_data=(x_val, y_val))
```

[336] ✓ 16.1s Python

```
# Plot
modified_model_11_history_dict = modified_model_11_history.history
modified_model_11_loss_values = modified_model_11_history_dict["loss"]
modified_model_11_val_loss_values = modified_model_11_history_dict["val_loss"]
modified_model_11_epochs = range(1, len(modified_model_11_loss_values) + 1)
modified_model_11_acc = modified_model_11_history_dict["accuracy"]
modified_model_11_val_acc = modified_model_11_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_11_epochs, modified_model_11_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_11_epochs, modified_model_11_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 11 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_11_epochs, modified_model_11_acc, "bo", label="Training acc")
ax2.plot(modified_model_11_epochs, modified_model_11_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 11 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[337] ✓ 0.1s Python



The optimal number of epochs for **Modified Model 11** is equal to 4. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 4.

The **Retrained Modified Model 11** has the test accuracy equals to 88.42% and the test loss value of 0.2885.

```
modified_model_R11 = keras.Sequential([
    layers.Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.000001)),
    layers.Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.000001)),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R11.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

modified_model_R11_epoch = 4

modified_model_R11_history = modified_model_R11.fit(partial_x_train,
                                    partial_y_train,
                                    epochs=modified_model_R11_epoch,
                                    batch_size=512,
                                    validation_data=(x_val, y_val))
```

[339] ✓ 7.4s Python

```
modified_model_R11_results = modified_model_R11.evaluate(x_test, y_test)
print("Retrained Modeified Model 11 - Test Accuracy: ", round(modified_model_R11_results[1],4))
print("Retrained Modifired Model 11 - Test Loss Value: ", round(modified_model_R11_results[0],4))
```

[340] ✓ 6.9s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.8837 - loss: 0.2895
Retrained Modeified Model 11 - Test Accuracy:  0.8842
Retrained Modifired Model 11 - Test Loss Value:  0.2885
```

7.2 Dropout

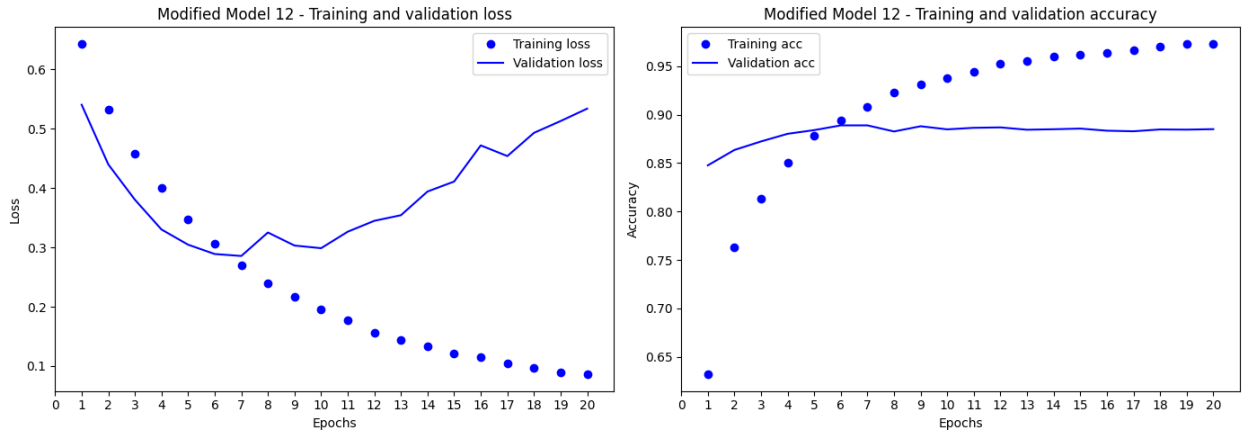
This model has 2 hidden layers of 16 nodes with ‘relu’ activation function, and a single node for the last layer with ‘Sigmoid’ activation function. I did the two-layers 50% dropout method to reduce the model complexity.

```
modified_model_12 = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(16, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])

modified_model_12.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

modified_model_12_history = modified_model_12.fit(partial_x_train,
                                    partial_y_train,
                                    epochs=20,
                                    batch_size=512,
                                    validation_data=(x_val, y_val))
```

[305] ✓ 14.8s Python

```python
# Plot
modified_model_12_history_dict = modified_model_12_history.history
modified_model_12_loss_values = modified_model_12_history_dict["loss"]
modified_model_12_val_loss_values = modified_model_12_history_dict["val_loss"]
modified_model_12_epochs = range(1, len(modified_model_12_loss_values) + 1)
modified_model_12_acc = modified_model_12_history_dict["accuracy"]
modified_model_12_val_acc = modified_model_12_history_dict["val_accuracy"]

# Create a figure with 1 row and 2 columns for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Plot for Loss
ax1.plot(modified_model_12_epochs, modified_model_12_loss_values, "bo", label="Training loss")
ax1.plot(modified_model_12_epochs, modified_model_12_val_loss_values, "b", label="Validation loss")
ax1.set_title("Modified Model 12 - Training and validation loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.set_xlim([0, 21])
ax1.set_xticks(range(0, 21, 1))
ax1.legend()

# Plot for Accuracy
ax2.plot(modified_model_12_epochs, modified_model_12_acc, "bo", label="Training acc")
ax2.plot(modified_model_12_epochs, modified_model_12_val_acc, "b", label="Validation acc")
ax2.set_title("Modified Model 12 - Training and validation accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.set_xlim([0, 21])
ax2.set_xticks(range(0, 21, 1))
ax2.legend()

# Show the combined figure
plt.tight_layout()
plt.show()
```

[306] ✓ 0.1s Python



The optimal number of epochs for **Modified Model 12** is equal to 7. As you can see the loss is at its bottom and the validation accuracy is at its top. Then, I rebuild the model with epoch equals to 7.

The **Retrained Modified Model 12** has the test accuracy equals to 88.25% and the test loss value of 0.288.

```
modified_model_R12 = keras.Sequential([
    layers.Dense(16, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(16, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])

modified_model_R12.compile(optimizer='rmsprop',
              loss = 'binary_crossentropy',
              metrics=['accuracy'])

modified_model_R12_epoch = 7

modified_model_R12_history = modified_model_R12.fit(partial_x_train,
                                    partial_y_train,
                                    epochs=modified_model_R12_epoch,
                                    batch_size=512,
                                    validation_data=(x_val, y_val))
```

[309] ✓ 9.8s Python

```
modified_model_R12_results = modified_model_R12.evaluate(x_test, y_test)
print("Retrained Modeified Model 12 - Test Accuracy: ", round(modified_model_R12_results[1],4))
print("Retrained Modifired Model 12 - Test Loss Value: ", round(modified_model_R12_results[0],4))
```

[310] ✓ 5.9s Python

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8822 - loss: 0.2875
Retrained Modeified Model 12 - Test Accuracy:  0.8825
Retrained Modifired Model 12 - Test Loss Value:  0.288
```