Advanced Machine Learning (BA-64061-001)

Assignment 2 – Neural Networks

Sgudise@kent.edu

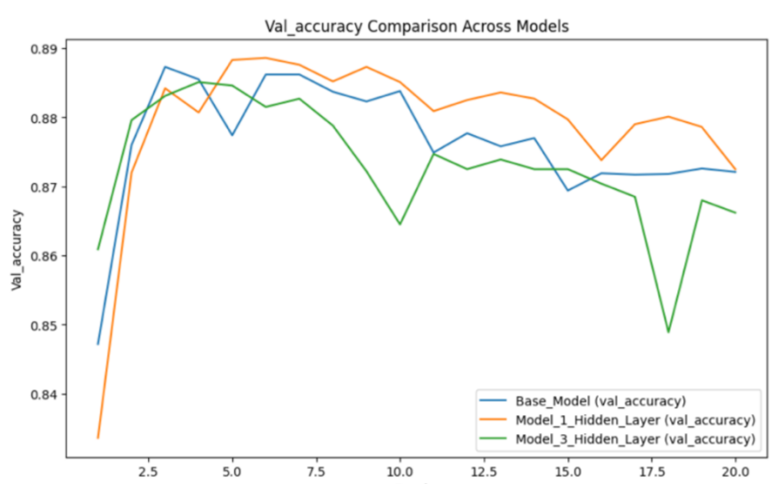
GitHub: Assignment 2 GitHub

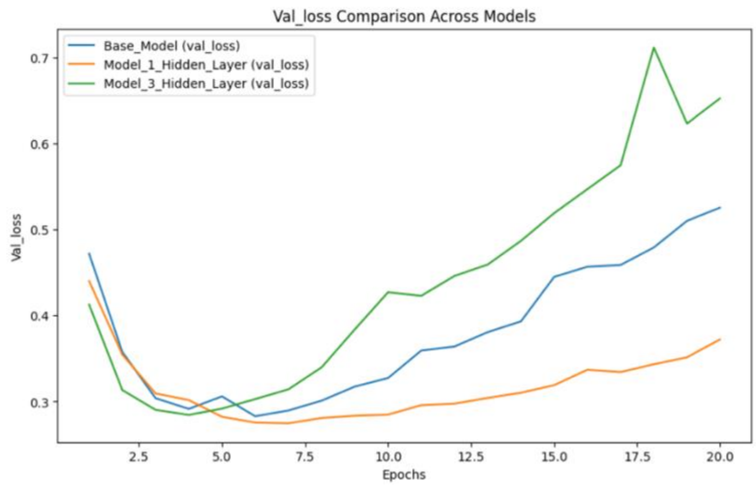
Q1. You used two hidden layers. Try using one or three hidden layers and see how doing so affects validation and test accuracy.

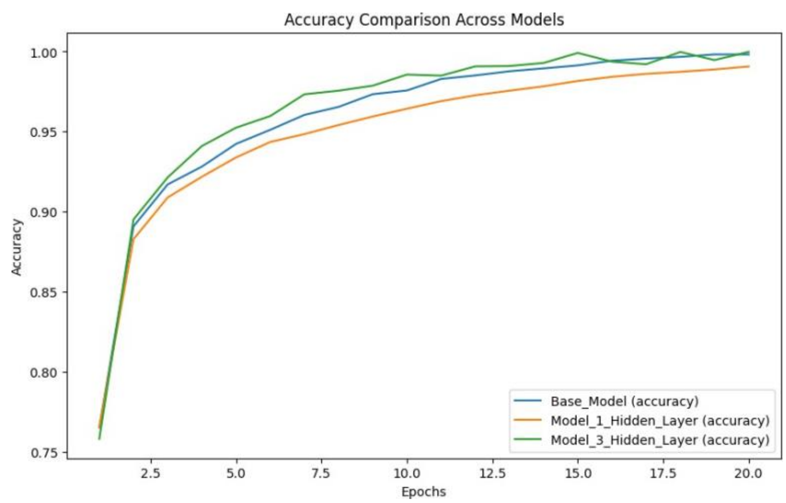
- The base model, which consists of two hidden layers, exhibits the highest validation accuracy and the lowest loss, surpassing both the single-layer and three-layer configurations.
- Model 1, incorporating a single hidden layer, demonstrates marginally reduced validation and training accuracy relative to the base model.
- Model 3, structured with three hidden layers, fails to yield performance improvements. Instead, it introduces instability, underscoring the fact that increasing the number of layers does not necessarily enhance model efficacy.
- The two-hidden-layer base model emerges as the most well-balanced and optimal configuration in terms of accuracy and stability.

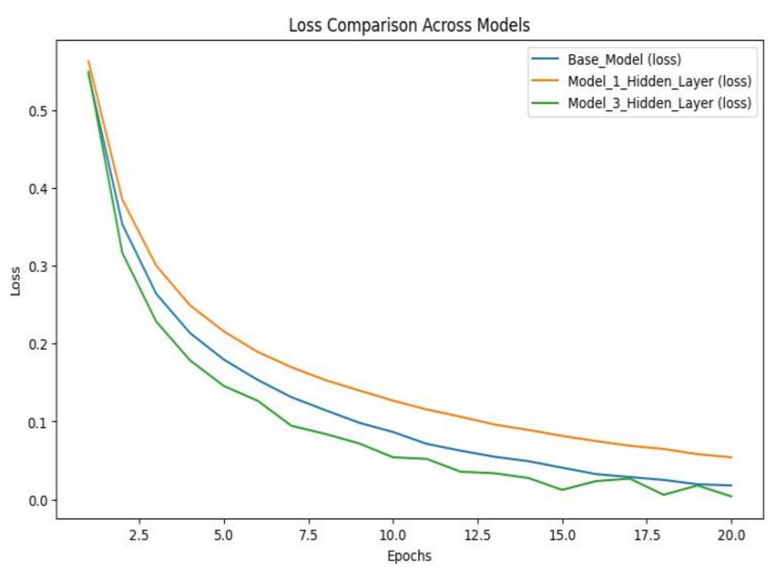
	Test loss	Test Accuracy
Base Model	0.28	0.88
1HL	0.28	0.88
3HL	0.33	0.87

***Note the code snipping's will be attached at the end of the report*




Val_loss Comparison Across Models
Base_Model (val_loss)
Model_1_Hidden_Layer (val_loss)
Model_3_Hidden_Layer (val_loss)
Val_loss
Epochs

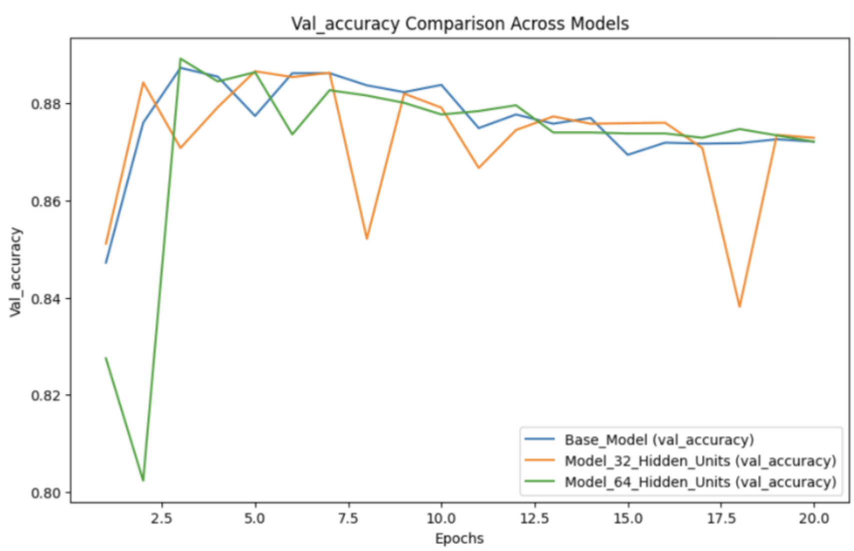

Accuracy Comparison Across Models
Base_Model (accuracy)
Model_1_Hidden_Layer (accuracy)
Model_3_Hidden_Layer (accuracy)
Accuracy
Epochs

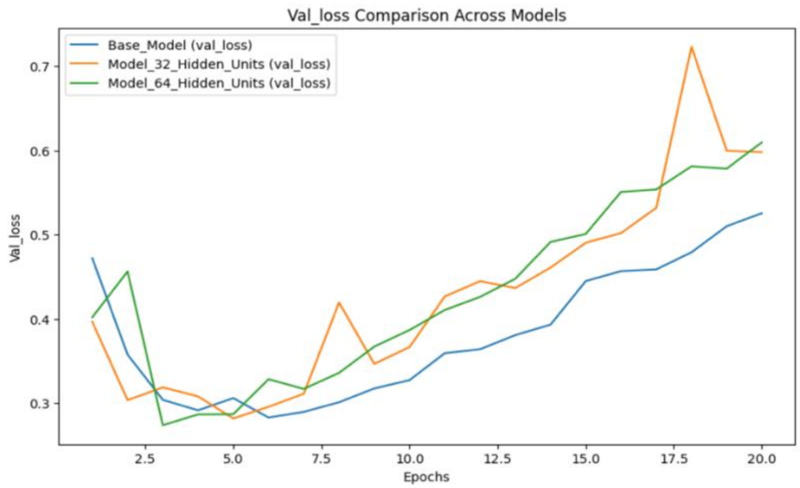

Loss Comparison Across Models
Base_Model (loss)
Model_1_Hidden_Layer (loss)
Model_3_Hidden_Layer (loss)
Loss
Epochs

Q2. Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.

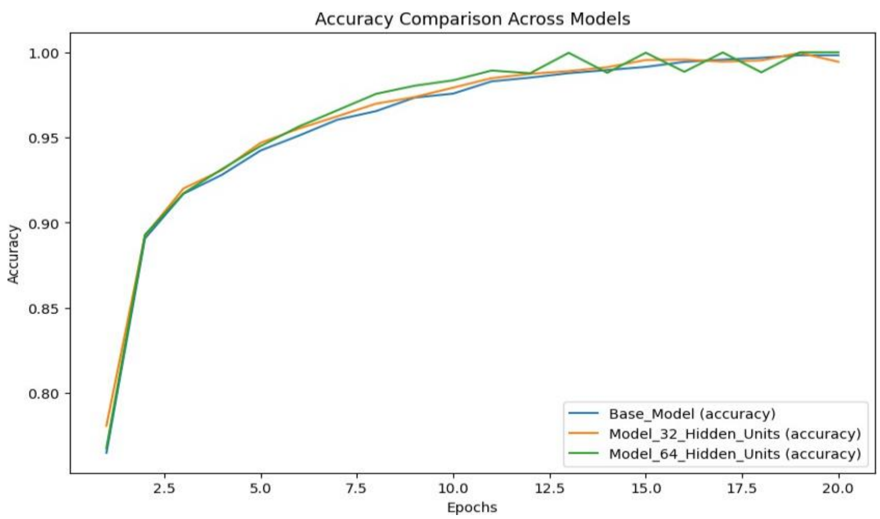
- In terms of validation accuracy, the model with 32 units outperforms those with 16 and 64 units. The 64-unit model exhibits lower and more fluctuating validation accuracy, while the 32-unit model performs similarly to the baseline 16-unit model.
- The model with 64 hidden units achieves the lowest validation loss compared to the 32 and 16-unit models, particularly at higher epochs. This suggests that increasing the number of hidden units may lead to overfitting and reduced generalization.
- All models with 16, 32, and 64 units show similar performance in training accuracy and loss, but the baseline model performs slightly better overall.
- In conclusion, increasing the number of hidden units from 16 to 32 to 64 does not significantly enhance the model's performance but introduces more volatility. Therefore, the baseline model remains the better choice.

	Test loss	**Test Accuracy**
Base Model	0.28	0.88
32HU	0.29	0.88
64HU	0.27	0.89

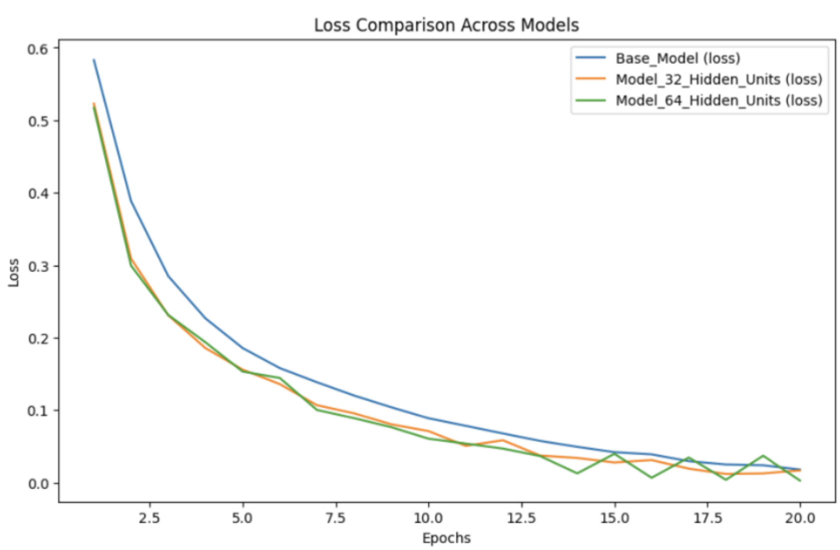




Val_loss Comparison Across Models
Base_Model (val_loss)
Model_32_Hidden_Units (val_loss)
Model_64_Hidden_Units (val_loss)
Val_loss
Epochs



Accuracy Comparison Across Models
Base_Model (accuracy)
Model_32_Hidden_Units (accuracy)
Model_64_Hidden_Units (accuracy)
Accuracy
Epochs

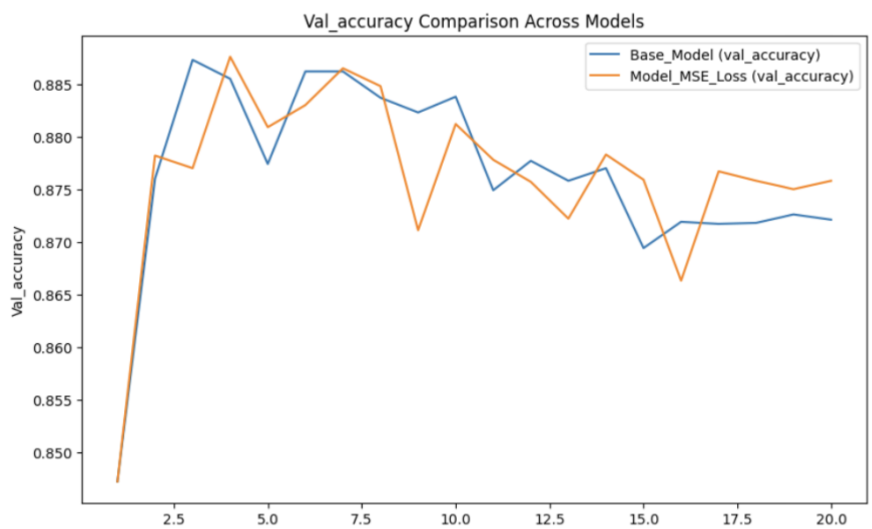


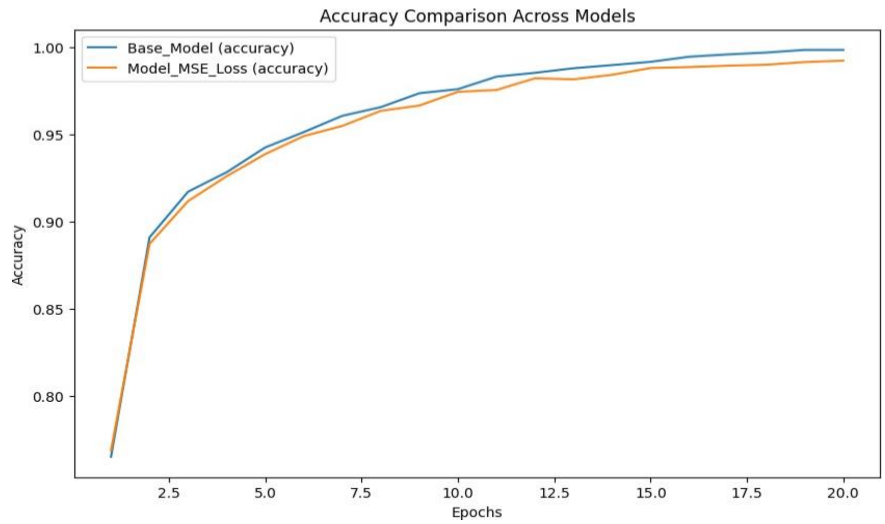
Loss Comparison Across Models
Base_Model (loss)
Model_32_Hidden_Units (loss)
Model_64_Hidden_Units (loss)
Loss
Epochs

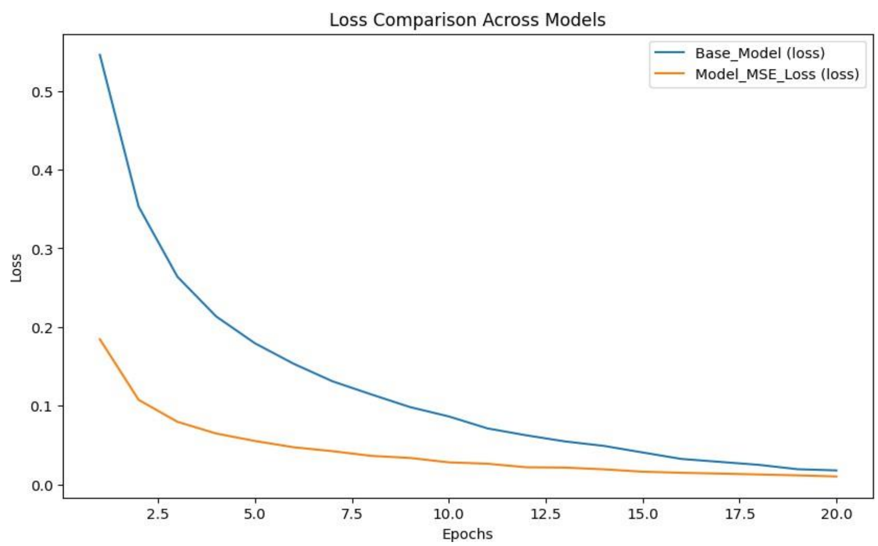
Q3. Try using the MSE loss function instead of binary cross entropy.

- BCE is more suitable for binary classification problems; however, in this case, MSE performs significantly better than BCE.
- MSE notably outperforms BCE in terms of validation loss, exhibiting a lower validation loss compared to the baseline model. This indicates that the MSE model more effectively reduces the error between predicted and actual values.
- The MSE model achieves better results with fewer training epochs.
- In conclusion, for this specific model, MSE is clearly the superior choice over BCE.

	Test loss	**Test Accuracy**
Base Model	0.28	0.88
MSE	0.08	0.88



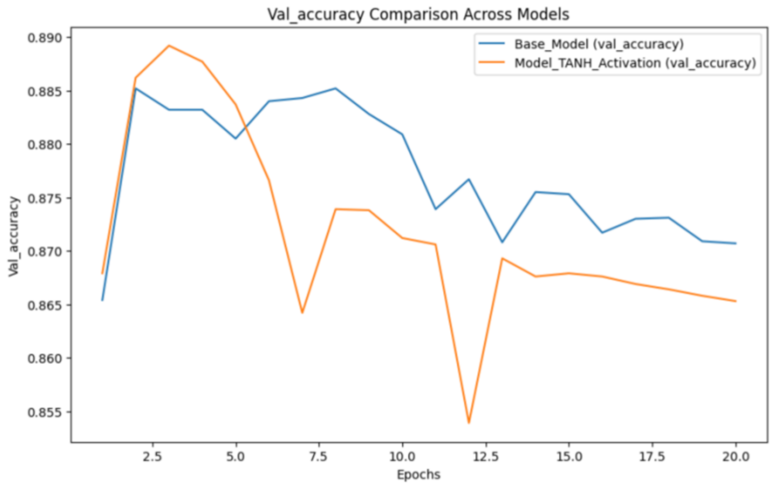


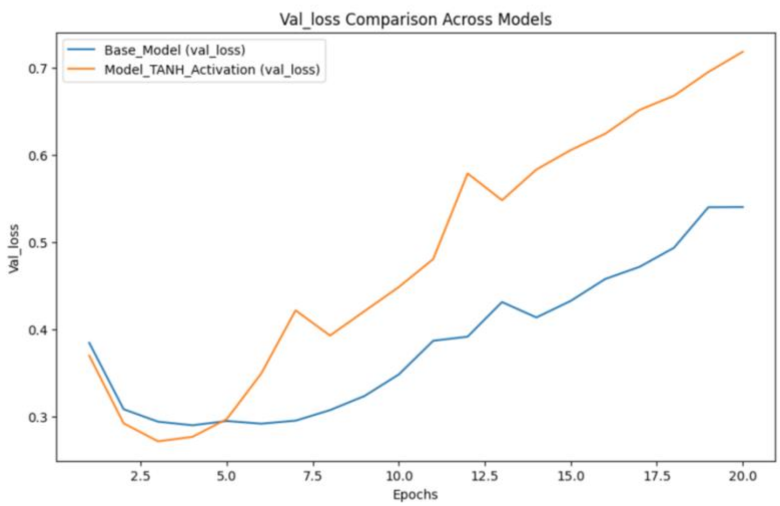


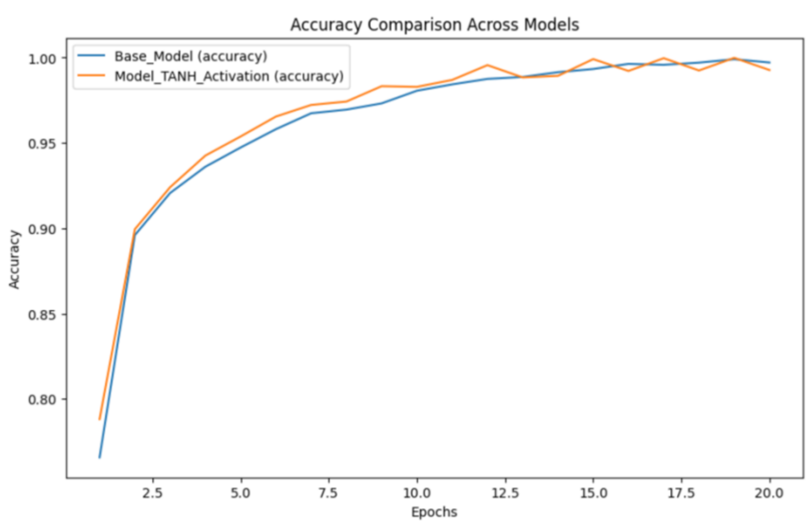
Q4. Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relul.

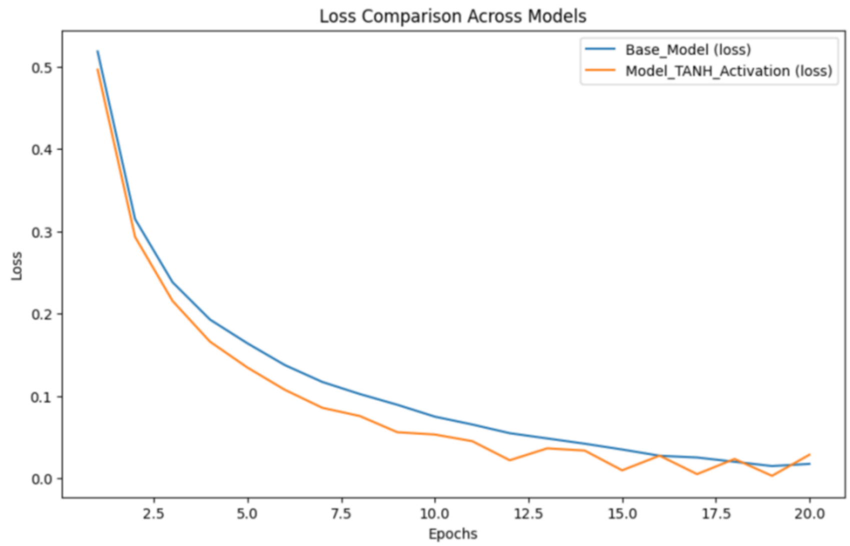
- The ReLU activation (baseline model) outperforms the tanh activation in terms of accuracy.
- The ReLU model exhibits a lower loss compared to the tanh model, indicating that it more effectively minimizes the error between predicted and actual values.
- In conclusion, ReLU is the better choice as it achieves higher accuracy and lower loss than tanh.

	Test loss	**Test Accuracy**
Base Model	0.28	0.88
Tanh	0.42	0.86


Val_accuracy Comparison Across Models
Base_Model (val_accuracy)
Model_TANH_Activation (val_accuracy)
Val_accuracy
Epochs
0.890
0.885
0.880
0.875
0.870
0.865
0.860
0.855
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Val_loss Comparison Across Models
Base_Model (val_loss)
Model_TANH_Activation (val_loss)
Val_loss
Epochs
0.7
0.6
0.5
0.4
0.3
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

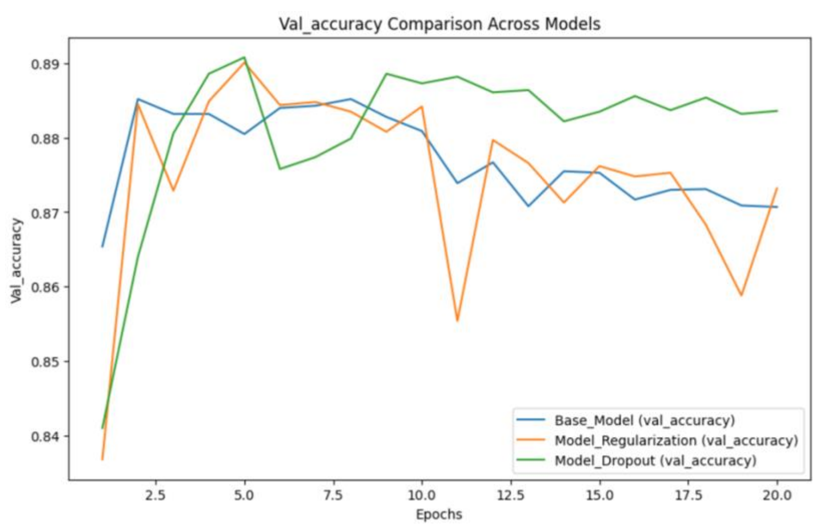

Accuracy Comparison Across Models
Base_Model (accuracy)
Model_TANH_Activation (accuracy)
Accuracy
Epochs
1.00
0.95
0.90
0.85
0.80
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

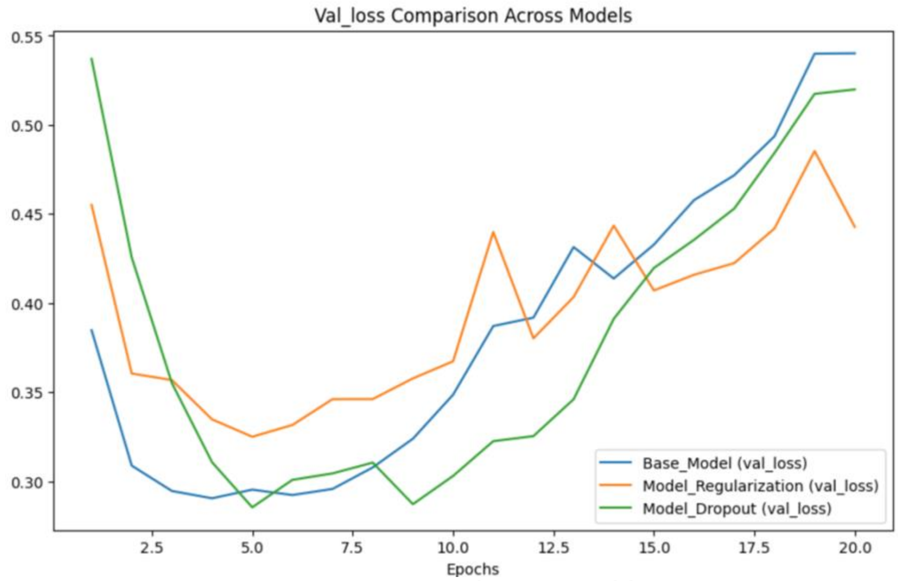


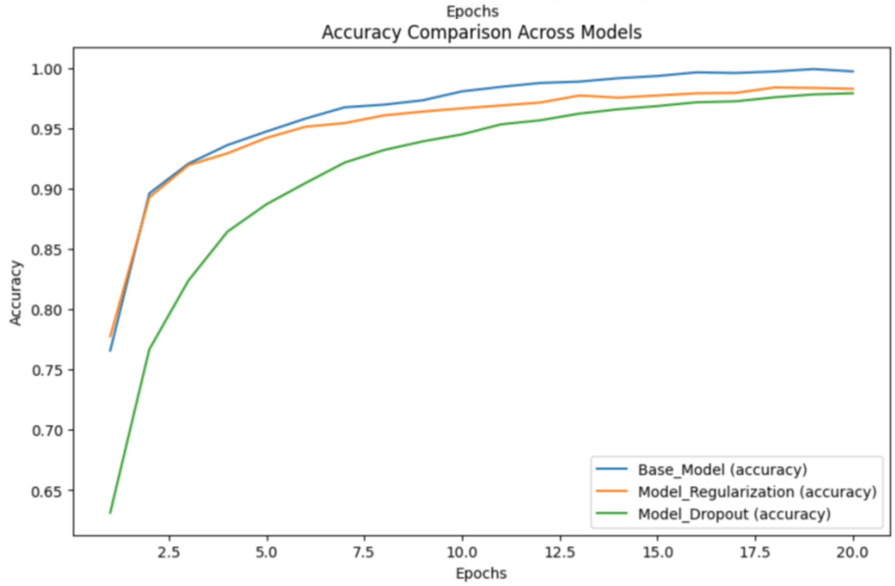
Q5. Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation

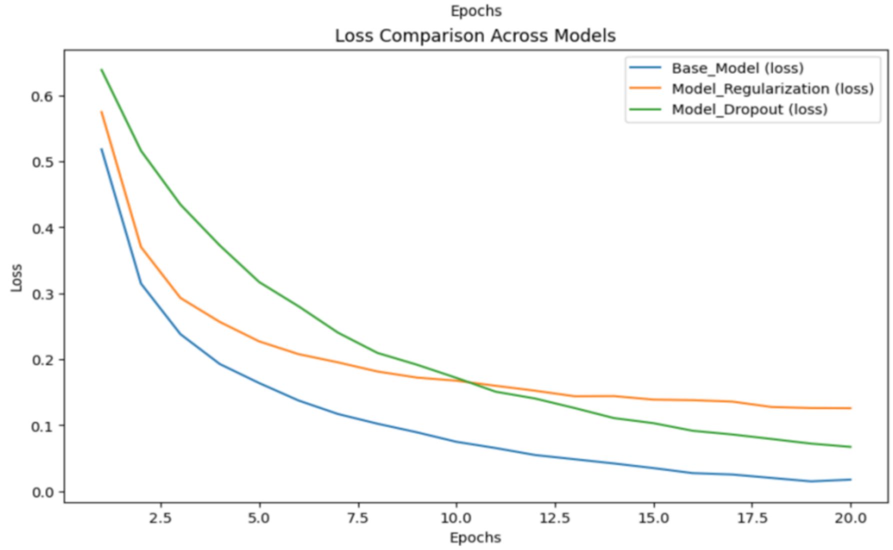
1. Dropout optimization achieves the highest accuracy compared to the baseline model and L2 regularization.
2. Dropout also performs best in terms of loss, having the lowest error rate, with L2 following closely behind, while the baseline model is the least effective.
3. Based on the overall performance across all graphs, dropout emerges as the most effective optimization method for this model, outperforming both the baseline and L2 regularization.

	Test loss	**Test Accuracy**
L2	0.34	0.88
Dropout	0.32	0.88




Val_loss Comparison Across Models
0.55
0.50
0.45
0.40
0.35
0.30
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs
Base_Model (val_loss)
Model_Regularization (val_loss)
Model_Dropout (val_loss)

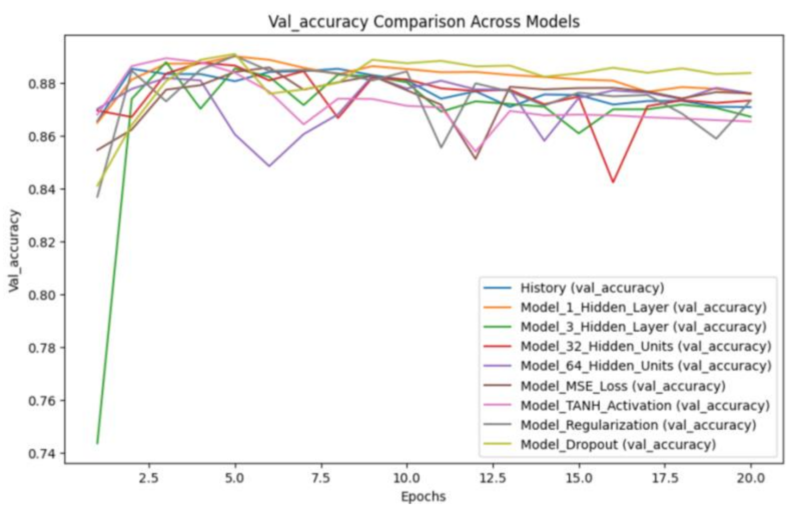

Accuracy Comparison Across Models
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs
Base_Model (accuracy)
Model_Regularization (accuracy)
Model_Dropout (accuracy)

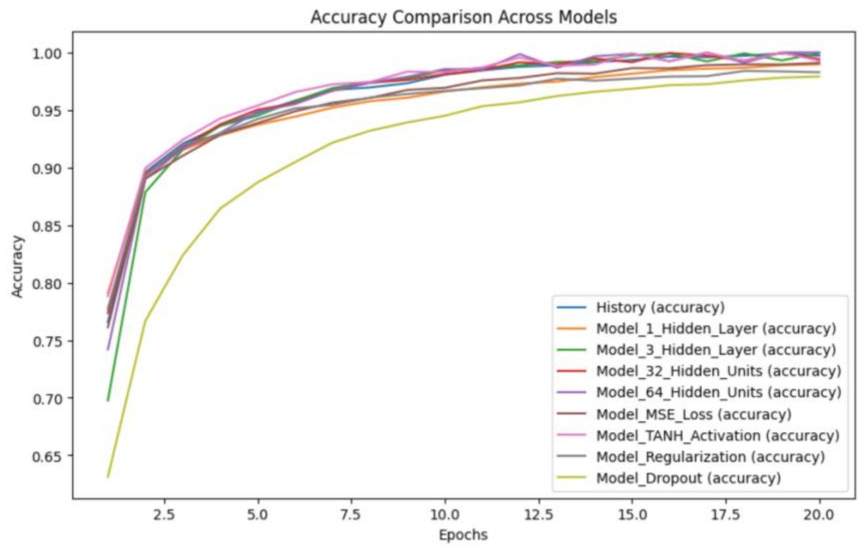

Loss Comparison Across Models
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs
Base_Model (loss)
Model_Regularization (loss)
Model_Dropout (loss)

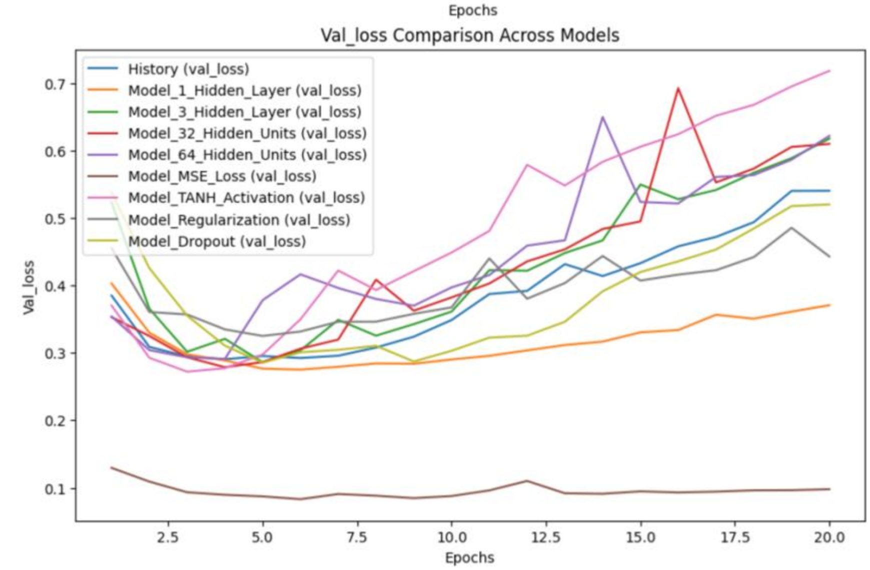
Comparing all the models:

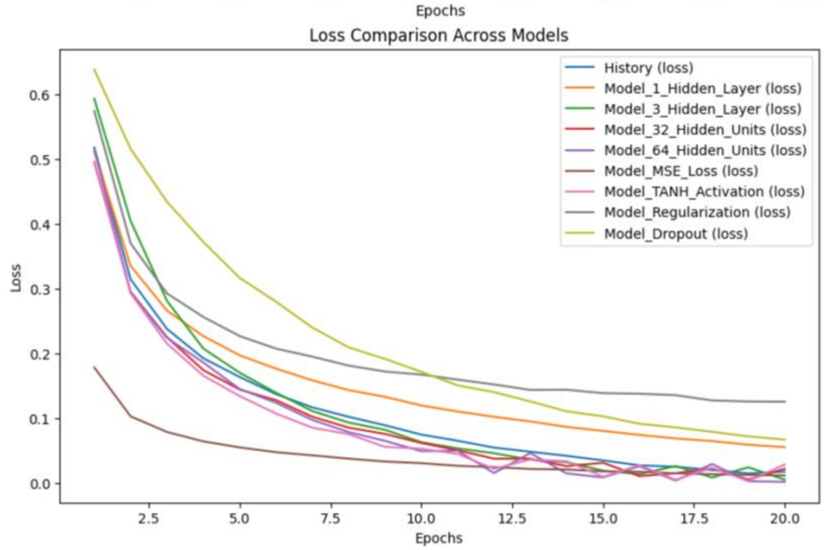
- Among all models, the one with 32 hidden units achieved the highest validation accuracy.
- In terms of accuracy, both the tanh model and the baseline model performed similarly and were the best in this aspect.
- When comparing validation loss and overall loss, the model with 64 hidden units outperformed the others.
- In conclusion, the model with 64 hidden units provided the best balance and performed well across all key metrics.

Model	Test Loss	Test Accuracy
Base Model	0.28	0.89
1HL	0.28	0.89
3HL	0.33	0.87
32HU	0.29	0.88
64HU	0.27	0.89
MSE	0.08	0.88
Tanh	0.42	0.88
L2	0.34	0.88
Dropout	0.32	0.88









*** Below are the code snippets printed from Google Collab, as they were too large for screenshots.

Loading the IMDB dataset

```
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
```

```
train_data[0]
```

```
 104,
 88,
 4,
 381,
 15,
 297,
 98,
 32,
 2071,
 56,
 26,
 141,
 6,
 194,
 7486,
 18,
 4,
 226,
 22,
 21,
 134,
 476,
 26,
 480,
 5,
 144,
 30,
 5535,
 18,
 51,
 36,
 28,
 224,
 92,
 25,
 104,
 4,
 226,
 65,
 16,
 38,
 1334,
 88,
 12,
 16,
 283,
 5,
 16,
 4472,
 113,
 103,
 32,
 15,
 16,
 5345,
 19,
 178,
 32]
```

```
train_labels[0]
```

```
1
```

```
max([max(sequence) for sequence in train_data])
```

```
9999
```

Decoding reviews back to text

```
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
```

Preparing the data

Encoding the integer sequences via multi-hot encoding

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

```
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")
```

Building your model differnt configurations

0. Model definition - Given by professor

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

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

1. Using mod_1_hid_lay to build the model with 1 hidden layer

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

mod_1_Hid_lay = keras.Sequential([
    layers.Dense(16, activation="relu"), # Building the model with 1 hidden layer
    layers.Dense(1, activation="sigmoid")
])

mod_1_Hid_lay.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

2. Using mod_3_hid_lay to construct the model with 3 hidden layers

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

```
mod_3_Hid_lay = keras.Sequential([
    layers.Dense(16, activation="relu"), # hidden layer 1
    layers.Dense(16, activation="relu"), # hidden layer 2
    layers.Dense(16, activation="relu"), # hidden layer 3
    layers.Dense(1, activation="sigmoid")
])

mod_3_Hid_lay.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

3. (Question 2) Constructing the model with 32 (mod_32_Hid_Units) fewer hidden units

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

mod_32_Hid_Units = keras.Sequential([
    layers.Dense(32, activation="relu"), # hidden units 32
    layers.Dense(32, activation="relu"), # hidden units 32
    layers.Dense(1, activation="sigmoid")
])

mod_32_Hid_Units.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

4. (Question 2) Building the model with higher hidden units 64 (mod_64_Hid_Units).

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

mod_64_Hid_Units = keras.Sequential([
    layers.Dense(64, activation="relu"), # hidden units 64
    layers.Dense(64, activation="relu"), # hidden units 64
    layers.Dense(1, activation="sigmoid")
])

mod_64_Hid_Units.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

5. (Question 3) Building the base model with mse loss function (model_mse_Loss)

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

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

mod_mse_Loss.compile(optimizer="rmsprop",
              loss="mse",
              metrics=["accuracy"])
```

6. (Question 4) Building the model with tanh activation

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

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

7. (Question 5) Building the model with regularization (mod_reg)

```
from tensorflow.keras import regularizers

mod_reg = 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")
])

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

8. (Question 5) Building the model with dropout (mod_drop)

```
mod_drop = 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")
])

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

Validating your approach

Setting aside a validation set

```
x_value = x_train[:10000]
x_partial_training = x_train[10000:]
y_value = y_train[:10000]
y_partial_training= y_train[10000:]
```

Training your model

```
history = model.fit(x_partial_training,
                    y_partial_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 68ms/step - accuracy: 0.6733 - loss: 0.5989 - val_accuracy: 0.8654 - val_loss: 0.3848
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8974 - loss: 0.3280 - val_accuracy: 0.8852 - val_loss: 0.3088
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9207 - loss: 0.2474 - val_accuracy: 0.8832 - val_loss: 0.2945
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9398 - loss: 0.1917 - val_accuracy: 0.8832 - val_loss: 0.2904
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9486 - loss: 0.1630 - val_accuracy: 0.8805 - val_loss: 0.2953
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9612 - loss: 0.1349 - val_accuracy: 0.8840 - val_loss: 0.2923
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9694 - loss: 0.1146 - val_accuracy: 0.8843 - val_loss: 0.2957
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9717 - loss: 0.1012 - val_accuracy: 0.8852 - val_loss: 0.3077
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9769 - loss: 0.0857 - val_accuracy: 0.8828 - val_loss: 0.3239
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9842 - loss: 0.0698 - val_accuracy: 0.8809 - val_loss: 0.3486
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9863 - loss: 0.0625 - val_accuracy: 0.8739 - val_loss: 0.3871
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9876 - loss: 0.0571 - val_accuracy: 0.8767 - val_loss: 0.3918
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9909 - loss: 0.0455 - val_accuracy: 0.8708 - val_loss: 0.4314
Epoch 14/20
```

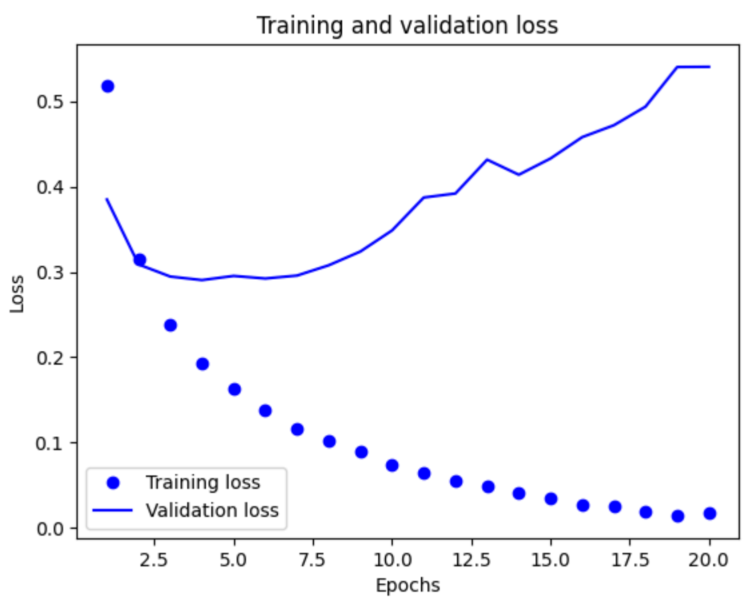
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9939 - loss: 0.0380 - val_accuracy: 0.8755 - val_loss: 0.4138
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9959 - loss: 0.0293 - val_accuracy: 0.8753 - val_loss: 0.4328
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9973 - loss: 0.0251 - val_accuracy: 0.8717 - val_loss: 0.4578
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.9973 - loss: 0.0212 - val_accuracy: 0.8730 - val_loss: 0.4717
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.9988 - loss: 0.0163 - val_accuracy: 0.8731 - val_loss: 0.4935
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9991 - loss: 0.0138 - val_accuracy: 0.8709 - val_loss: 0.5399
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9988 - loss: 0.0139 - val_accuracy: 0.8707 - val_loss: 0.5401
```

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

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

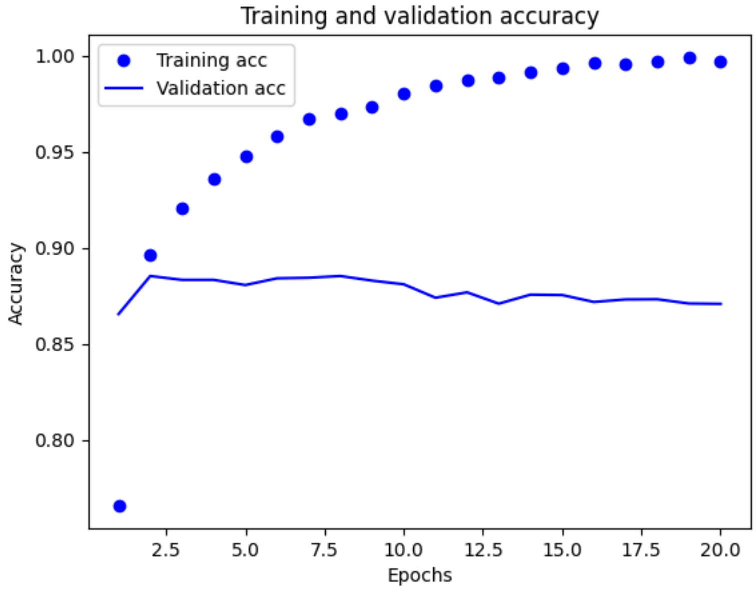
Plotting the training and validation loss

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



Plotting the training and validation accuracy

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



Retraining a model from scratch

```
model = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.7334 - loss: 0.5733
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.8997 - loss: 0.2980
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9218 - loss: 0.2228
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9338 - loss: 0.1861
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8836 - loss: 0.2855
```

```
results
```

```
[0.2851715087890625, 0.8855599761009216]
```

Using a trained model to generate predictions on new data

```
model.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.17808114],
       [0.9998711 ],
       [0.64469516],
       ...,
       [0.08423662],
       [0.06379367],
       [0.52848387]], dtype=float32)
```

Further experiments

1. Model With 1 Hidden Layer

```
Model_Hid_lay_1 = mod_1_Hid_lay.fit(x_partial_training,
                    y_partial_training,
```

```
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

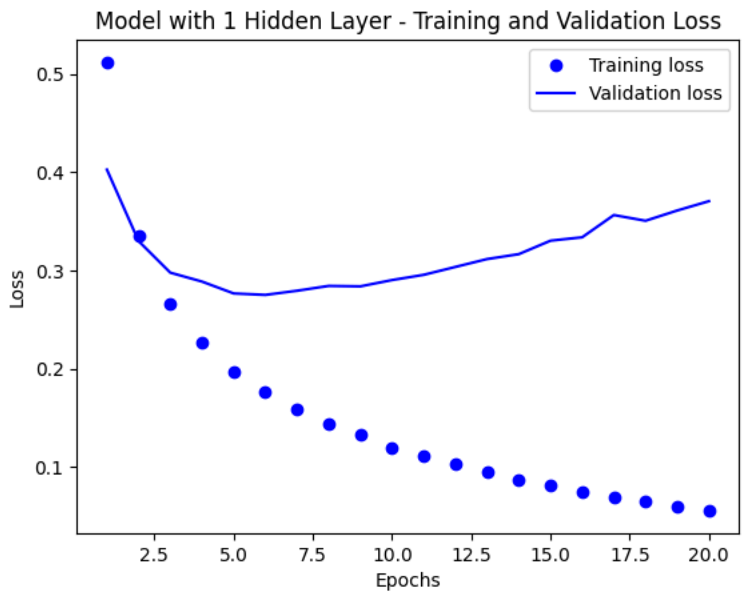
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.7100 - loss: 0.5866 - val_accuracy: 0.8648 - val_loss: 0.4027
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.8937 - loss: 0.3503 - val_accuracy: 0.8811 - val_loss: 0.3299
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9178 - loss: 0.2722 - val_accuracy: 0.8871 - val_loss: 0.2977
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9309 - loss: 0.2274 - val_accuracy: 0.8872 - val_loss: 0.2886
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9408 - loss: 0.1966 - val_accuracy: 0.8899 - val_loss: 0.2766
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9457 - loss: 0.1770 - val_accuracy: 0.8886 - val_loss: 0.2751
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9564 - loss: 0.1546 - val_accuracy: 0.8856 - val_loss: 0.2793
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9616 - loss: 0.1404 - val_accuracy: 0.8831 - val_loss: 0.2842
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9636 - loss: 0.1301 - val_accuracy: 0.8862 - val_loss: 0.2838
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9692 - loss: 0.1169 - val_accuracy: 0.8851 - val_loss: 0.2902
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9751 - loss: 0.1061 - val_accuracy: 0.8839 - val_loss: 0.2955
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.9751 - loss: 0.1002 - val_accuracy: 0.8840 - val_loss: 0.3036
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9763 - loss: 0.0929 - val_accuracy: 0.8829 - val_loss: 0.3116
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9784 - loss: 0.0865 - val_accuracy: 0.8821 - val_loss: 0.3166
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9826 - loss: 0.0787 - val_accuracy: 0.8812 - val_loss: 0.3303
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9871 - loss: 0.0726 - val_accuracy: 0.8807 - val_loss: 0.3337
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9873 - loss: 0.0676 - val_accuracy: 0.8765 - val_loss: 0.3564
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9883 - loss: 0.0610 - val_accuracy: 0.8783 - val_loss: 0.3506
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9901 - loss: 0.0569 - val_accuracy: 0.8776 - val_loss: 0.3610
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9918 - loss: 0.0522 - val_accuracy: 0.8756 - val_loss: 0.3705
```

```
Mod_1_Hid_Lay_dict = Model_Hid_lay_1.history
Mod_1_Hid_Lay_dict.keys()
```

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

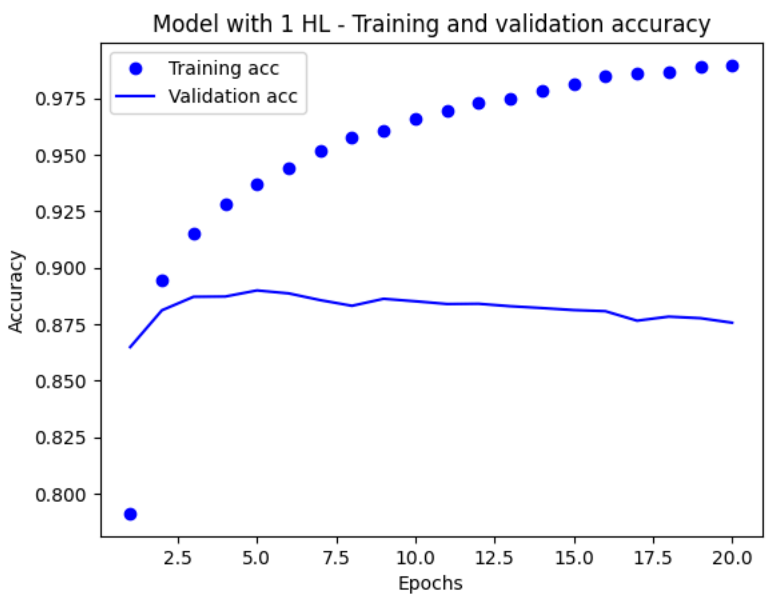
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Mod_1_Hid_Lay_dict = Model_Hid_lay_1.history
loss_values_1 = Mod_1_Hid_Lay_dict["loss"]  # Training loss values
val_loss_values_1 = Mod_1_Hid_Lay_dict["val_loss"]  # Validation loss values

epochs = range(1, len(loss_values_1) + 1)

# Plot the loss values
plt.plot(epochs, loss_values_1, "bo", label="Training loss")  # Training loss with blue dots
plt.plot(epochs, val_loss_values_1, "b", label="Validation loss")  # Validation loss with blue line
plt.title("Model with 1 Hidden Layer - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

```



Plotting Accuracy

```
plt.clf()
acc_1 = Mod_1_Hid_Lay_dict["accuracy"]
val_acc_1 = Mod_1_Hid_Lay_dict["val_accuracy"]
plt.plot(epochs, acc_1, "bo", label="Training acc")
plt.plot(epochs, val_acc_1, "b", label="Validation acc")
plt.title("Model with 1 HL - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()

```



Retraining

```
mod_1_Hid_lay = keras.Sequential([
    layers.Dense(16, activation="relu"),  # 1 Hidden Layer
    layers.Dense(1, activation="sigmoid")
])

mod_1_Hid_lay.compile(optimizer="rmsprop",
                   loss="binary_crossentropy",
                   metrics=["accuracy"])
```

```

# Train the model
mod_1_Hid_lay.fit(x_train, y_train, epochs=4, batch_size=512)

# Evaluate the model
model_1_HL_results = mod_1_Hid_lay.evaluate(x_test, y_test)  # Consistent naming

```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.7468 - loss: 0.5295
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9069 - loss: 0.2872
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9210 - loss: 0.2284
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9341 - loss: 0.1969
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8870 - loss: 0.2780
```

```
model_1_HL_results

```

```
[0.2771202623844147, 0.8882799744606018]
```

Using Trained data to predict

```
mod_1_Hid_lay.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.21187341],
       [0.9994522 ],
       [0.7810599 ],
       ...,
       [0.11715537],
       [0.10660435],
       [0.47352076]], dtype=float32)
```

2. Model With 3 Hidden Layer

```
Model_3_Hid_Lay = mod_3_Hid_lay.fit(x_partial_training,
                    y_partial_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 0.6063 - loss: 0.6408 - val_accuracy: 0.7437 - val_loss: 0.5233
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8627 - loss: 0.4395 - val_accuracy: 0.8738 - val_loss: 0.3654
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9174 - loss: 0.2930 - val_accuracy: 0.8877 - val_loss: 0.3015
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9356 - loss: 0.2176 - val_accuracy: 0.8701 - val_loss: 0.3207
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9497 - loss: 0.1646 - val_accuracy: 0.8853 - val_loss: 0.2861
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9667 - loss: 0.1293 - val_accuracy: 0.8822 - val_loss: 0.3036
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9730 - loss: 0.1059 - val_accuracy: 0.8715 - val_loss: 0.3489
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9726 - loss: 0.0959 - val_accuracy: 0.8830 - val_loss: 0.3254
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9775 - loss: 0.0823 - val_accuracy: 0.8823 - val_loss: 0.3425
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9864 - loss: 0.0611 - val_accuracy: 0.8801 - val_loss: 0.3611
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9866 - loss: 0.0536 - val_accuracy: 0.8690 - val_loss: 0.4225
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9906 - loss: 0.0431 - val_accuracy: 0.8729 - val_loss: 0.4216
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9939 - loss: 0.0297 - val_accuracy: 0.8719 - val_loss: 0.4479
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9947 - loss: 0.0271 - val_accuracy: 0.8709 - val_loss: 0.4666
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9983 - loss: 0.0173 - val_accuracy: 0.8608 - val_loss: 0.5493
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9984 - loss: 0.0169 - val_accuracy: 0.8699 - val_loss: 0.5276
Epoch 17/20
```

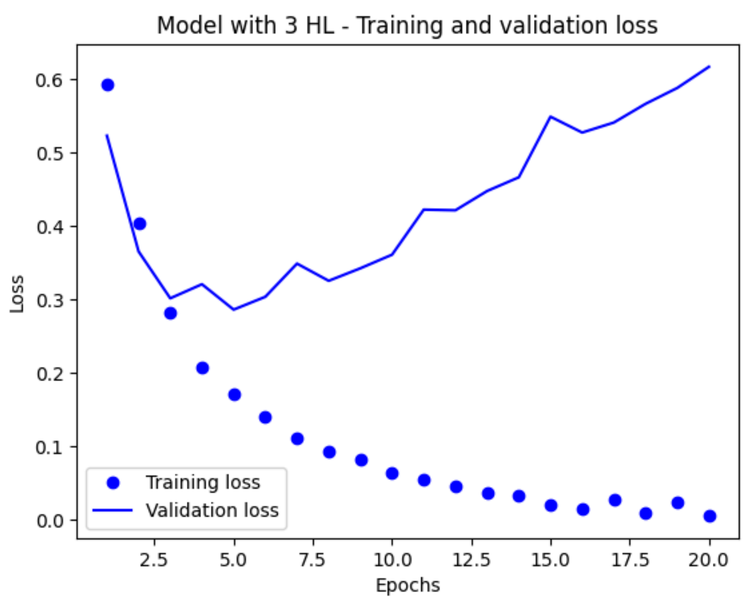
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9903 - loss: 0.0302 - val_accuracy: 0.8699 - val_loss: 0.5413
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9989 - loss: 0.0098 - val_accuracy: 0.8717 - val_loss: 0.5668
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9923 - loss: 0.0265 - val_accuracy: 0.8704 - val_loss: 0.5885
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9995 - loss: 0.0050 - val_accuracy: 0.8671 - val_loss: 0.6173
```

```
Model_3_Hid_Lay_dict = Model_3_Hid_Lay.history
Model_3_Hid_Lay_dict.keys()
```

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

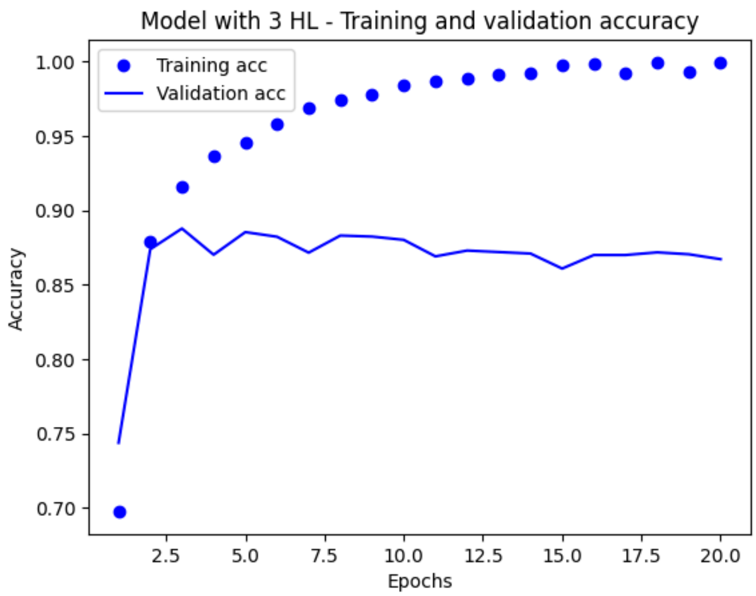
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_3_Hid_Lay_dict = Model_3_Hid_Lay.history
loss_values_3 = Model_3_Hid_Lay_dict["loss"]
val_loss_values_3 = Model_3_Hid_Lay_dict["val_loss"]
epochs = range(1, len(loss_values_3) + 1)
plt.plot(epochs, loss_values_3, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_3, "b", label="Validation loss")
plt.title("Model with 3 HL - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
acc_3 = Model_3_Hid_Lay_dict["accuracy"]
val_acc_3 = Model_3_Hid_Lay_dict["val_accuracy"]
plt.plot(epochs, acc_3, "bo", label="Training acc")
plt.plot(epochs, val_acc_3, "b", label="Validation acc")
plt.title("Model with 3 HL - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining

```
mod_3_Hid_lay = keras.Sequential([
    layers.Dense(16, activation="relu"), # 1 Hidden Layer
    layers.Dense(16, activation="relu"), # 2 Hidden Layer
    layers.Dense(16, activation="relu"), # 3 Hidden Layer
    layers.Dense(1, activation="sigmoid")
])
mod_3_Hid_lay.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
mod_3_Hid_lay.fit(x_train, y_train, epochs=6, batch_size=512) # Epochs selected 6 because it starts to dip from 7
Model_3_Hid_Lay_Results = mod_3_Hid_lay.evaluate(x_test, y_test)
```

```
Epoch 1/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - accuracy: 0.7227 - loss: 0.5586
Epoch 2/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9066 - loss: 0.2763
Epoch 3/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9234 - loss: 0.2102
Epoch 4/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9422 - loss: 0.1668
Epoch 5/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9505 - loss: 0.1410
Epoch 6/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9561 - loss: 0.1255
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8701 - loss: 0.3586
```

```
Model_3_Hid_Lay_Results
```

```
[0.3565736711025238, 0.8722000122070312]
```

Using Trained data to predict

```
mod_3_Hid_lay.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.08351237],
       [0.99981767],
       [0.10274448],
       ...,
       [0.04394579],
       [0.03569042],
       [0.53605026]], dtype=float32)
```

3. Model With 32 Hidden Units

```
Mod_32_Hid_Units = mod_32_Hid_Units.fit(x_partial_training,
                    y_partial_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

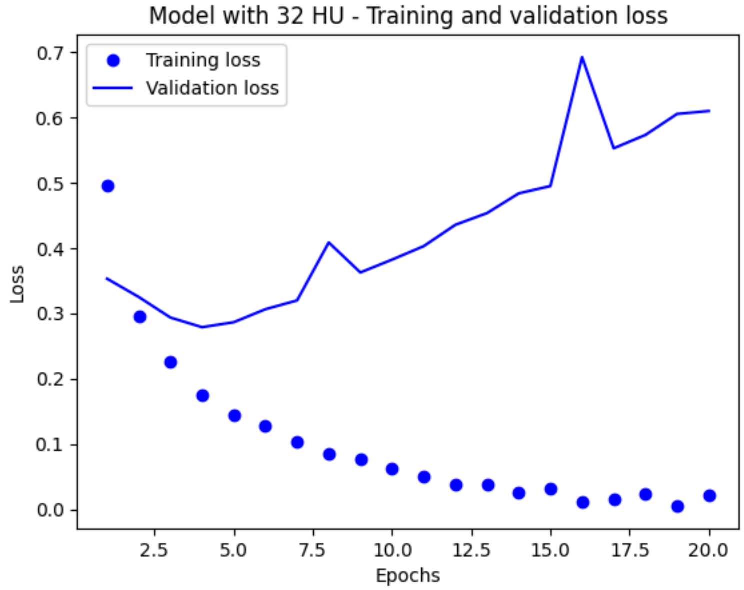
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 66ms/step - accuracy: 0.6927 - loss: 0.5789 - val_accuracy: 0.8694 - val_loss: 0.3529
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.8870 - loss: 0.3142 - val_accuracy: 0.8670 - val_loss: 0.3249
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9209 - loss: 0.2275 - val_accuracy: 0.8834 - val_loss: 0.2934
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9398 - loss: 0.1742 - val_accuracy: 0.8875 - val_loss: 0.2786
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9510 - loss: 0.1455 - val_accuracy: 0.8864 - val_loss: 0.2862
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - accuracy: 0.9581 - loss: 0.1245 - val_accuracy: 0.8808 - val_loss: 0.3061
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9725 - loss: 0.0942 - val_accuracy: 0.8843 - val_loss: 0.3196
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9779 - loss: 0.0792 - val_accuracy: 0.8666 - val_loss: 0.4084
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9785 - loss: 0.0712 - val_accuracy: 0.8818 - val_loss: 0.3624
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9856 - loss: 0.0530 - val_accuracy: 0.8812 - val_loss: 0.3821
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9886 - loss: 0.0432 - val_accuracy: 0.8778 - val_loss: 0.4027
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9944 - loss: 0.0325 - val_accuracy: 0.8768 - val_loss: 0.4354
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9922 - loss: 0.0326 - val_accuracy: 0.8773 - val_loss: 0.4533
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9974 - loss: 0.0205 - val_accuracy: 0.8718 - val_loss: 0.4835
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9972 - loss: 0.0186 - val_accuracy: 0.8747 - val_loss: 0.4947
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9997 - loss: 0.0102 - val_accuracy: 0.8423 - val_loss: 0.6922
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9929 - loss: 0.0268 - val_accuracy: 0.8711 - val_loss: 0.5525
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9942 - loss: 0.0194 - val_accuracy: 0.8733 - val_loss: 0.5729
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9997 - loss: 0.0055 - val_accuracy: 0.8723 - val_loss: 0.6051
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9931 - loss: 0.0251 - val_accuracy: 0.8732 - val_loss: 0.6096
```

```
Mod_32_Hid_Units_dict = Mod_32_Hid_Units.history
Mod_32_Hid_Units_dict.keys()
```

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

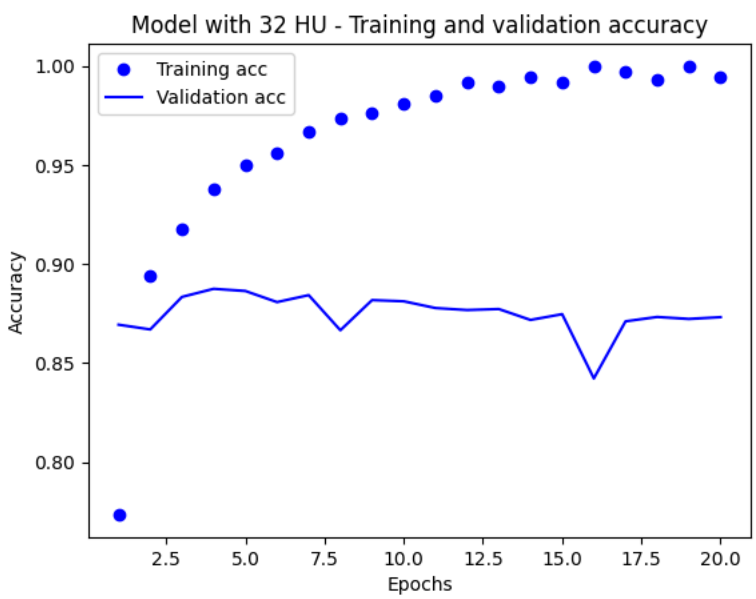
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Mod_32_Hid_Units_dict = Mod_32_Hid_Units.history
loss_values_32 = Mod_32_Hid_Units_dict["loss"]
val_loss_values_32 = Mod_32_Hid_Units_dict["val_loss"]
epochs = range(1, len(loss_values_32) + 1)
plt.plot(epochs, loss_values_32, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_32, "b", label="Validation loss")
plt.title("Model with 32 HU - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
acc_32 = Mod_32_Hid_Units_dict["accuracy"]
val_acc_32 = Mod_32_Hid_Units_dict["val_accuracy"]
plt.plot(epochs, acc_32, "bo", label="Training acc")
plt.plot(epochs, val_acc_32, "b", label="Validation acc")
plt.title("Model with 32 HU - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining

```
mod_32_Hid_Units = keras.Sequential([
    layers.Dense(32, activation="relu"), # 32 Hidden Units
    layers.Dense(32, activation="relu"), # 32 Hidden Units
    layers.Dense(1, activation="sigmoid")
])
mod_32_Hid_Units.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

```
mod_32_Hid_Units.fit(x_train, y_train, epochs=3, batch_size=512) # Epochs selected 3 because it starts to dip from 3
Mod_32_Hid_Units_Results = mod_32_Hid_Units.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 20ms/step - accuracy: 0.7151 - loss: 0.5487
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9051 - loss: 0.2704
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9237 - loss: 0.2090
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8819 - loss: 0.2868
```

```
Mod_32_Hid_Units_Results
```

```
[0.2873370051383972, 0.8832799792289734]
```

Using Trained data to predict

```
mod_32_Hid_Units.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.16207792],
       [0.99952143],
       [0.662475  ],
       ...,
       [0.09712453],
       [0.07055076],
       [0.47576696]], dtype=float32)
```

4. Model With 64 Hidden Units

```
Mod_64_Hid_Units = mod_64_Hid_Units.fit(x_partial_training,
                    y_partial_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

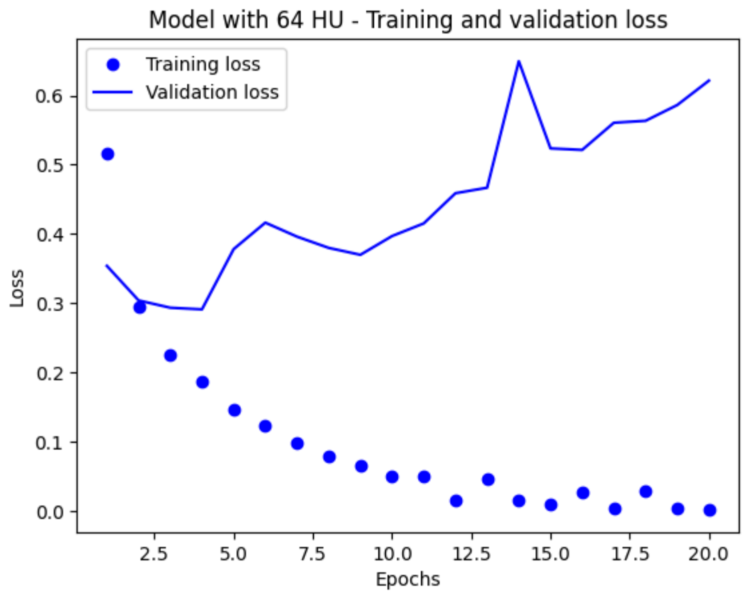
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - accuracy: 0.6499 - loss: 0.5961 - val_accuracy: 0.8700 - val_loss: 0.3538
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.8888 - loss: 0.3063 - val_accuracy: 0.8776 - val_loss: 0.3039
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9137 - loss: 0.2306 - val_accuracy: 0.8817 - val_loss: 0.2934
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9313 - loss: 0.1880 - val_accuracy: 0.8808 - val_loss: 0.2910
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9496 - loss: 0.1424 - val_accuracy: 0.8604 - val_loss: 0.3780
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9531 - loss: 0.1265 - val_accuracy: 0.8484 - val_loss: 0.4163
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9640 - loss: 0.1058 - val_accuracy: 0.8605 - val_loss: 0.3960
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.9705 - loss: 0.0850 - val_accuracy: 0.8680 - val_loss: 0.3798
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9798 - loss: 0.0651 - val_accuracy: 0.8825 - val_loss: 0.3698
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 57ms/step - accuracy: 0.9862 - loss: 0.0477 - val_accuracy: 0.8777 - val_loss: 0.3969
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9889 - loss: 0.0411 - val_accuracy: 0.8807 - val_loss: 0.4152
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.9987 - loss: 0.0150 - val_accuracy: 0.8774 - val_loss: 0.4588
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9794 - loss: 0.0656 - val_accuracy: 0.8772 - val_loss: 0.4667
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9992 - loss: 0.0091 - val_accuracy: 0.8580 - val_loss: 0.6494
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9958 - loss: 0.0157 - val_accuracy: 0.8742 - val_loss: 0.5235
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9975 - loss: 0.0113 - val_accuracy: 0.8770 - val_loss: 0.5213
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9999 - loss: 0.0039 - val_accuracy: 0.8762 - val_loss: 0.5606
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9988 - loss: 0.0061 - val_accuracy: 0.8737 - val_loss: 0.5634
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 53ms/step - accuracy: 1.0000 - loss: 0.0031 - val_accuracy: 0.8780 - val_loss: 0.5862
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 1.0000 - loss: 0.0019 - val_accuracy: 0.8759 - val_loss: 0.6214
```

```
Mod_64_Hid_Units_dict = Mod_64_Hid_Units.history
Mod_64_Hid_Units_dict.keys()
```

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

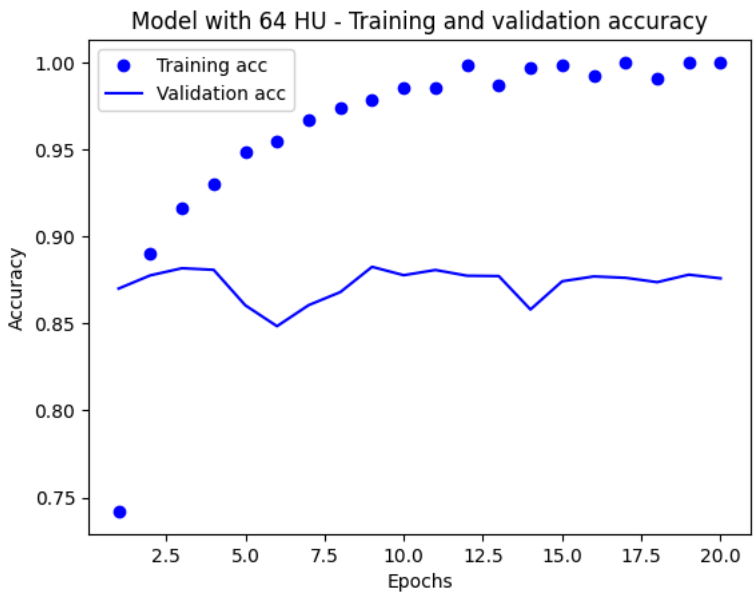
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Mod_64_Hid_Units_dict = Mod_64_Hid_Units.history
loss_values_64 = Mod_64_Hid_Units_dict["loss"]
val_loss_values_64 = Mod_64_Hid_Units_dict["val_loss"]
epochs = range(1, len(loss_values_64) + 1)
plt.plot(epochs, loss_values_64, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_64, "b", label="Validation loss")
plt.title("Model with 64 HU - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
acc_64 = Mod_64_Hid_Units_dict["accuracy"]
val_acc_64 = Mod_64_Hid_Units_dict["val_accuracy"]
plt.plot(epochs, acc_64, "bo", label="Training acc")
plt.plot(epochs, val_acc_64, "b", label="Validation acc")
plt.title("Model with 64 HU - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining

```
mod_64_Hid_Units = keras.Sequential([
    layers.Dense(64, activation="relu"), # 64 Hidden Units
    layers.Dense(64, activation="relu"), # 64 Hidden Units
    layers.Dense(1, activation="sigmoid")
])
mod_64_Hid_Units.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
mod_64_Hid_Units.fit(x_train, y_train, epochs=2, batch_size=512) # Epochs selected 2 because it starts to dip from 2
Mod_64_Hid_Units_Results = mod_64_Hid_Units.evaluate(x_test, y_test)
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.7186 - loss: 0.5402
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9027 - loss: 0.2596
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8491 - loss: 0.3573
```

```
Mod_64_Hid_Units_Results
```

```
[0.361043781042099, 0.8481600284576416]
```

```
mod_64_Hid_Units.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
array([[0.18302031],
       [0.99719894],
       [0.51645267],
       ...,
       [0.0594679 ],
       [0.04088284],
       [0.16301496]], dtype=float32)
```

5. Model With MSE Loss

```
Mod_MSE_LOSS = mod_mse_Loss.fit(x_partial_training,
                    y_partial_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - accuracy: 0.6722 - loss: 0.2094 - val_accuracy: 0.8545 - val_loss: 0.1295
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.8892 - loss: 0.1087 - val_accuracy: 0.8622 - val_loss: 0.1091
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9074 - loss: 0.0819 - val_accuracy: 0.8773 - val_loss: 0.0933
Epoch 4/20
```

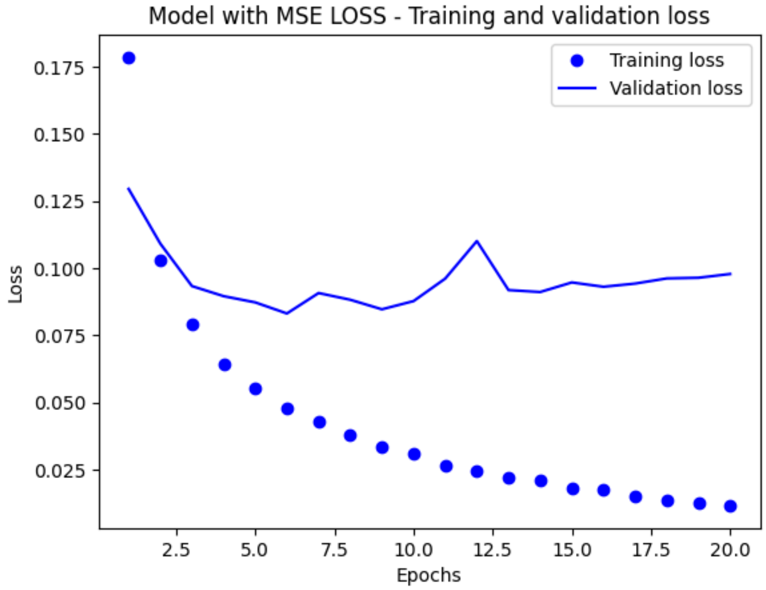
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9283 - loss: 0.0652 - val_accuracy: 0.8790 - val_loss: 0.0896
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9433 - loss: 0.0541 - val_accuracy: 0.8840 - val_loss: 0.0872
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9523 - loss: 0.0465 - val_accuracy: 0.8857 - val_loss: 0.0831
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9577 - loss: 0.0425 - val_accuracy: 0.8773 - val_loss: 0.0907
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9633 - loss: 0.0364 - val_accuracy: 0.8801 - val_loss: 0.0882
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9700 - loss: 0.0318 - val_accuracy: 0.8825 - val_loss: 0.0847
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9742 - loss: 0.0279 - val_accuracy: 0.8771 - val_loss: 0.0877
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9763 - loss: 0.0266 - val_accuracy: 0.8716 - val_loss: 0.0961
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9790 - loss: 0.0237 - val_accuracy: 0.8511 - val_loss: 0.1101
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9791 - loss: 0.0237 - val_accuracy: 0.8784 - val_loss: 0.0918
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9841 - loss: 0.0198 - val_accuracy: 0.8774 - val_loss: 0.0911
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9876 - loss: 0.0170 - val_accuracy: 0.8779 - val_loss: 0.0947
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9874 - loss: 0.0167 - val_accuracy: 0.8780 - val_loss: 0.0931
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9905 - loss: 0.0138 - val_accuracy: 0.8766 - val_loss: 0.0943
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9906 - loss: 0.0125 - val_accuracy: 0.8740 - val_loss: 0.0962
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9903 - loss: 0.0121 - val_accuracy: 0.8764 - val_loss: 0.0964
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9925 - loss: 0.0100 - val_accuracy: 0.8759 - val_loss: 0.0978
```

```python
Mod_MSE_LOSS_dict = Mod_MSE_LOSS.history
Mod_MSE_LOSS_dict.keys()
```

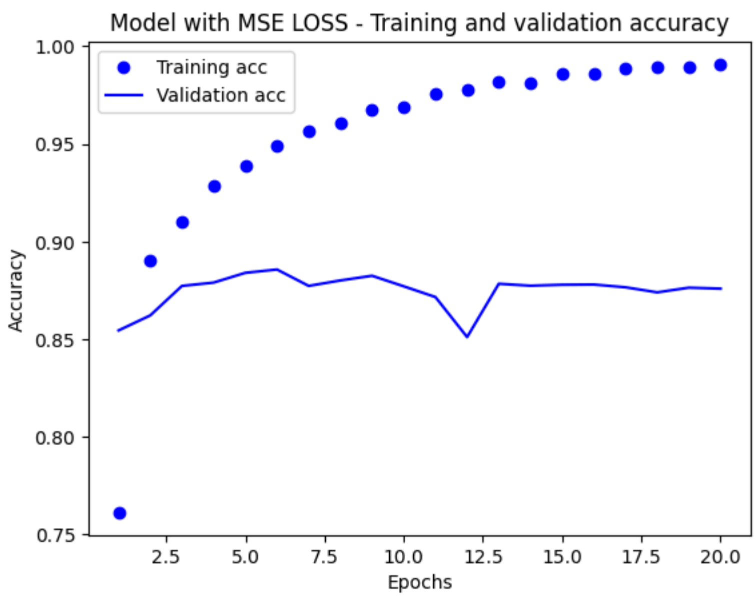
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

Plotting the graphshowing training and validation loss

```python
import matplotlib.pyplot as plt
Mod_MSE_LOSS_dict = Mod_MSE_LOSS.history
loss_values_MSE = Mod_MSE_LOSS_dict["loss"]
val_loss_values_MSE = Mod_MSE_LOSS_dict["val_loss"]
epochs = range(1, len(loss_values_MSE) + 1)
plt.plot(epochs, loss_values_MSE, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_MSE, "b", label="Validation loss")
plt.title("Model with MSE LOSS - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
plt.clf()
acc_MSE = Mod_MSE_LOSS_dict["accuracy"]
val_acc_MSE = Mod_MSE_LOSS_dict["val_accuracy"]
plt.plot(epochs, acc_MSE, "bo", label="Training acc")
plt.plot(epochs, val_acc_MSE, "b", label="Validation acc")
plt.title("Model with MSE LOSS - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining

```
mod_mse_Loss = keras.Sequential([
    layers.Dense(16, activation="relu"),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid")
])
mod_mse_Loss.compile(optimizer="rmsprop",
              loss="mse", # MSE Loss Function
              metrics=["accuracy"])
mod_mse_Loss.fit(x_train, y_train, epochs=4, batch_size=512) # Epochs selected 2 because it starts to dip from 2
Mod_MSE_LOSS_Results = mod_mse_Loss.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.6785 - loss: 0.2095
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.8975 - loss: 0.0952
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - accuracy: 0.9208 - loss: 0.0693
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9329 - loss: 0.0580
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8687 - loss: 0.0970
```

```
Mod_MSE_LOSS_Results
```

```
[0.09723840653896332, 0.8677600026130676]
```

```
mod_mse_Loss.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[0.17265368],
       [0.99926704],
       [0.3075071 ],
       ...,
       [0.10305677],
       [0.07930221],
       [0.31346497]], dtype=float32)
```

6. Model With tanh activation

```
Model_TANH_ACT = mod_tanh_act.fit(x_partial_training,
                    y_partial_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

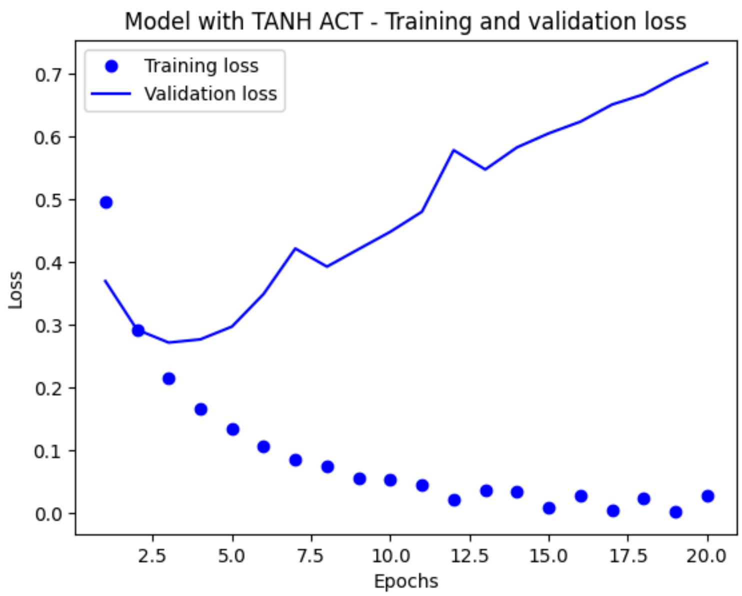
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.7051 - loss: 0.5749 - val_accuracy: 0.8679 - val_loss: 0.3701
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.8990 - loss: 0.3122 - val_accuracy: 0.8862 - val_loss: 0.2927
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9292 - loss: 0.2175 - val_accuracy: 0.8892 - val_loss: 0.2720
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9480 - loss: 0.1593 - val_accuracy: 0.8877 - val_loss: 0.2771
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9599 - loss: 0.1253 - val_accuracy: 0.8837 - val_loss: 0.2975
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9681 - loss: 0.1004 - val_accuracy: 0.8766 - val_loss: 0.3496
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9723 - loss: 0.0880 - val_accuracy: 0.8642 - val_loss: 0.4219
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9765 - loss: 0.0736 - val_accuracy: 0.8739 - val_loss: 0.3930
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9879 - loss: 0.0479 - val_accuracy: 0.8738 - val_loss: 0.4210
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9881 - loss: 0.0407 - val_accuracy: 0.8712 - val_loss: 0.4486
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9945 - loss: 0.0276 - val_accuracy: 0.8706 - val_loss: 0.4806
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9974 - loss: 0.0182 - val_accuracy: 0.8539 - val_loss: 0.5786
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9916 - loss: 0.0303 - val_accuracy: 0.8693 - val_loss: 0.5479
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9977 - loss: 0.0133 - val_accuracy: 0.8676 - val_loss: 0.5831
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9992 - loss: 0.0094 - val_accuracy: 0.8679 - val_loss: 0.6054
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9966 - loss: 0.0150 - val_accuracy: 0.8676 - val_loss: 0.6240
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9997 - loss: 0.0050 - val_accuracy: 0.8669 - val_loss: 0.6513
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9939 - loss: 0.0195 - val_accuracy: 0.8664 - val_loss: 0.6675
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 1.0000 - loss: 0.0029 - val_accuracy: 0.8658 - val_loss: 0.6947
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9967 - loss: 0.0138 - val_accuracy: 0.8653 - val_loss: 0.7178
```

```
Mod_TANH_ACT_dict = Model_TANH_ACT.history
Mod_TANH_ACT_dict.keys()
```

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

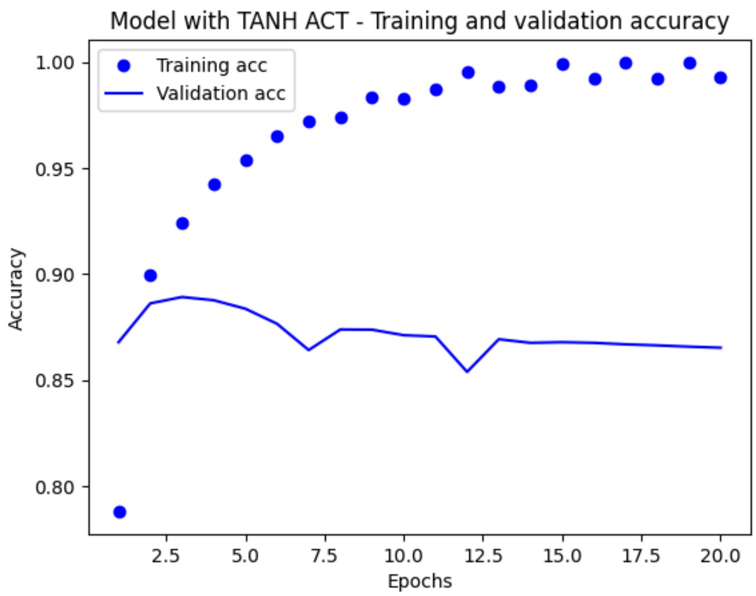
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Mod_TANH_ACT_dict = Model_TANH_ACT.history
loss_values_TANH = Mod_TANH_ACT_dict["loss"]
val_loss_values_TANH = Mod_TANH_ACT_dict["val_loss"]
epochs = range(1, len(loss_values_TANH) + 1)
plt.plot(epochs, loss_values_TANH, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_TANH, "b", label="Validation loss")
plt.title("Model with TANH ACT - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
acc_TANH = Mod_TANH_ACT_dict["accuracy"]
val_acc_TANH = Mod_TANH_ACT_dict["val_accuracy"]
plt.plot(epochs, acc_TANH, "bo", label="Training acc")
plt.plot(epochs, val_acc_TANH, "b", label="Validation acc")
plt.title("Model with TANH ACT - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining

```
model_tanh_act = keras.Sequential([
    layers.Dense(16, activation="tanh"), # tanh activation
    layers.Dense(16, activation="tanh"), # tanh activation
    layers.Dense(1, activation="sigmoid")
])
mod_tanh_act.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
mod_tanh_act.fit(x_train, y_train, epochs=3, batch_size=512) # Epochs selected 3 because it starts to dip from 3
Model_TANH_ACT_Results = mod_tanh_act.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9382 - loss: 0.3110
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9603 - loss: 0.1403
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - accuracy: 0.9687 - loss: 0.1024
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8587 - loss: 0.4397
```

```
Model_TANH_ACT_Results
```

```
[0.4309842884540558, 0.8623999953269958]
```

```
model_tanh_act.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.49555385],
       [0.46224463],
       [0.4193119 ],
       ...,
       [0.52889866],
       [0.5111811 ],
       [0.48294097]], dtype=float32)
```

7. Model With L2 Regularization

```
Mod_Reg_Tech = mod_reg.fit(x_partial_training,
                    y_partial_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - accuracy: 0.6930 - loss: 0.6505 - val_accuracy: 0.8368 - val_loss: 0.4551
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8887 - loss: 0.3883 - val_accuracy: 0.8845 - val_loss: 0.3605
Epoch 3/20
```

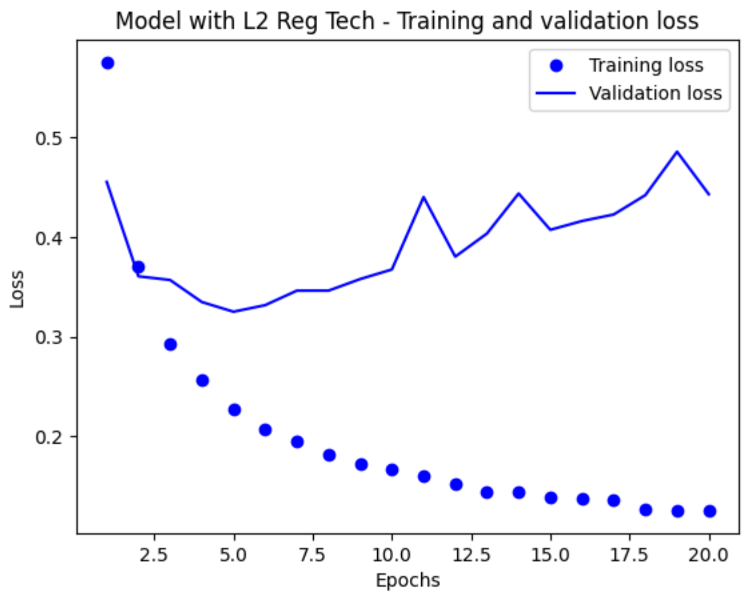
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9195 - loss: 0.2978 - val_accuracy: 0.8729 - val_loss: 0.3568
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9326 - loss: 0.2545 - val_accuracy: 0.8849 - val_loss: 0.3347
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9480 - loss: 0.2204 - val_accuracy: 0.8901 - val_loss: 0.3250
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9589 - loss: 0.2002 - val_accuracy: 0.8844 - val_loss: 0.3315
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9597 - loss: 0.1890 - val_accuracy: 0.8848 - val_loss: 0.3461
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9645 - loss: 0.1771 - val_accuracy: 0.8835 - val_loss: 0.3461
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9702 - loss: 0.1632 - val_accuracy: 0.8808 - val_loss: 0.3577
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9717 - loss: 0.1600 - val_accuracy: 0.8842 - val_loss: 0.3673
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9720 - loss: 0.1528 - val_accuracy: 0.8554 - val_loss: 0.4399
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9699 - loss: 0.1535 - val_accuracy: 0.8797 - val_loss: 0.3802
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9797 - loss: 0.1400 - val_accuracy: 0.8766 - val_loss: 0.4033
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9794 - loss: 0.1367 - val_accuracy: 0.8713 - val_loss: 0.4434
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9796 - loss: 0.1370 - val_accuracy: 0.8762 - val_loss: 0.4071
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9838 - loss: 0.1303 - val_accuracy: 0.8748 - val_loss: 0.4159
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9863 - loss: 0.1245 - val_accuracy: 0.8753 - val_loss: 0.4224
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9884 - loss: 0.1191 - val_accuracy: 0.8683 - val_loss: 0.4418
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9869 - loss: 0.1217 - val_accuracy: 0.8588 - val_loss: 0.4853
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9873 - loss: 0.1185 - val_accuracy: 0.8732 - val_loss: 0.4427
```

```
Mod_Reg_Tech_dict = Mod_Reg_Tech.history
Mod_Reg_Tech_dict.keys()
```

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

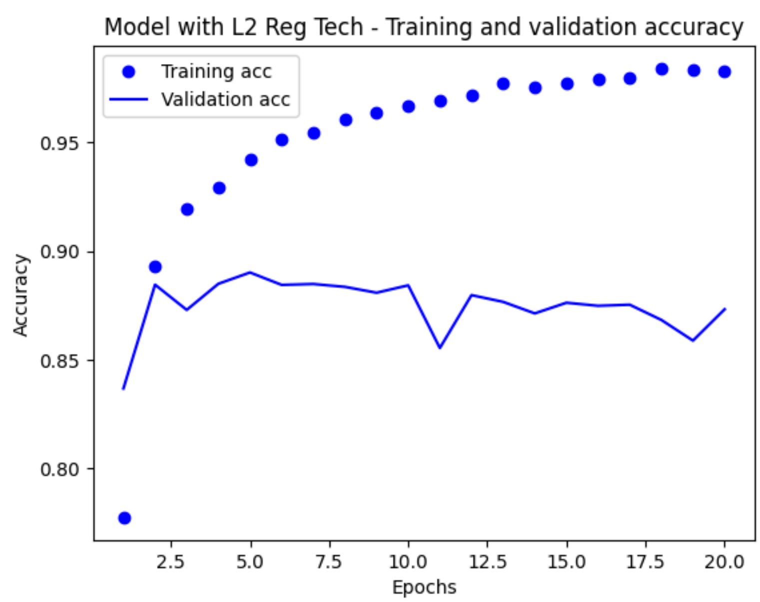
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Mod_Reg_Tech_dict = Mod_Reg_Tech.history
loss_values_Reg = Mod_Reg_Tech_dict["loss"]
val_loss_values_Reg = Mod_Reg_Tech_dict["val_loss"]
epochs = range(1, len(loss_values_Reg) + 1)
plt.plot(epochs, loss_values_Reg, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_Reg, "b", label="Validation loss")
plt.title("Model with L2 Reg Tech - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
acc_Reg = Mod_Reg_Tech_dict["accuracy"]
val_acc_Reg = Mod_Reg_Tech_dict["val_accuracy"]
plt.plot(epochs, acc_Reg, "bo", label="Training acc")
plt.plot(epochs, val_acc_Reg, "b", label="Validation acc")
plt.title("Model with L2 Reg Tech - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining

```
mod_reg = keras.Sequential([
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)), # Applied L2 regularization (0.001 - com
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)), # Applied L2 regularization (0.001 - com
    layers.Dense(1, activation="sigmoid")
])
mod_reg.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

```
mod_reg.fit(x_train, y_train, epochs=2, batch_size=512) # Epochs selected 2 because it starts to dip from 3
Mod_Reg_Tech_Results = mod_reg.evaluate(x_test, y_test)
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - accuracy: 0.7367 - loss: 0.6064
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9012 - loss: 0.3421
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8864 - loss: 0.3397
```

```
Mod_Reg_Tech_Results
```

```
[0.33807697892189026, 0.8866000175476074]
```

```
mod_reg.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.31154406],
       [0.9974691 ],
       [0.86966914],
       ...,
       [0.16900912],
       [0.16153361],
       [0.56198907]], dtype=float32)
```

8. Model With Dropout Technique`

```
Mod_Drp_Tech = mod_drop.fit(x_partial_training,
                    y_partial_training,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_value, y_value))
```

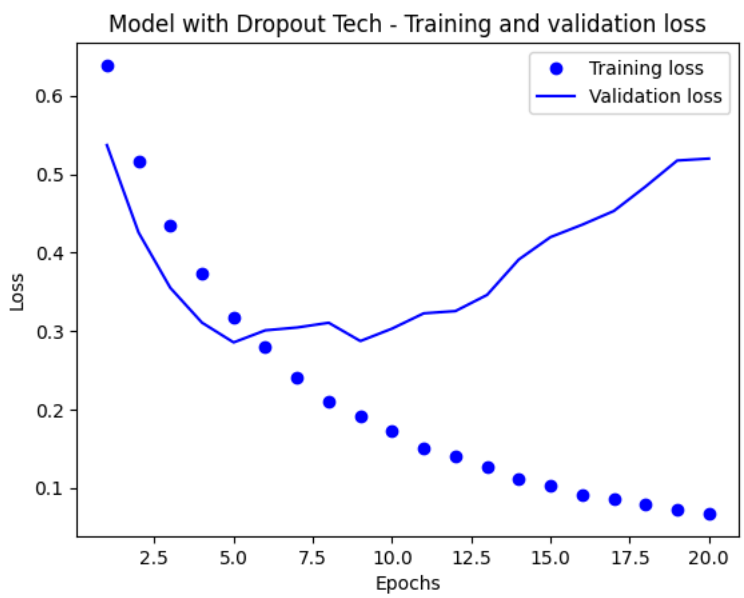
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 0.5822 - loss: 0.6665 - val_accuracy: 0.8410 - val_loss: 0.5370
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.7552 - loss: 0.5348 - val_accuracy: 0.8640 - val_loss: 0.4255
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.8175 - loss: 0.4446 - val_accuracy: 0.8806 - val_loss: 0.3551
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8595 - loss: 0.3833 - val_accuracy: 0.8886 - val_loss: 0.3106
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.8880 - loss: 0.3216 - val_accuracy: 0.8908 - val_loss: 0.2854
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9021 - loss: 0.2841 - val_accuracy: 0.8758 - val_loss: 0.3008
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9182 - loss: 0.2476 - val_accuracy: 0.8774 - val_loss: 0.3044
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9310 - loss: 0.2149 - val_accuracy: 0.8799 - val_loss: 0.3104
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9369 - loss: 0.1945 - val_accuracy: 0.8886 - val_loss: 0.2871
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9488 - loss: 0.1667 - val_accuracy: 0.8873 - val_loss: 0.3030
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9544 - loss: 0.1454 - val_accuracy: 0.8882 - val_loss: 0.3225
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9592 - loss: 0.1358 - val_accuracy: 0.8861 - val_loss: 0.3253
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9629 - loss: 0.1248 - val_accuracy: 0.8864 - val_loss: 0.3460
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9660 - loss: 0.1114 - val_accuracy: 0.8822 - val_loss: 0.3911
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9699 - loss: 0.1024 - val_accuracy: 0.8835 - val_loss: 0.4197
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9702 - loss: 0.0931 - val_accuracy: 0.8856 - val_loss: 0.4355
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.9724 - loss: 0.0849 - val_accuracy: 0.8837 - val_loss: 0.4530
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.9773 - loss: 0.0743 - val_accuracy: 0.8854 - val_loss: 0.4842
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9763 - loss: 0.0767 - val_accuracy: 0.8832 - val_loss: 0.5174
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9798 - loss: 0.0633 - val_accuracy: 0.8836 - val_loss: 0.5198
```

```
Mod_Drp_Tech_dict = Mod_Drp_Tech.history
Mod_Drp_Tech_dict.keys()
```

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

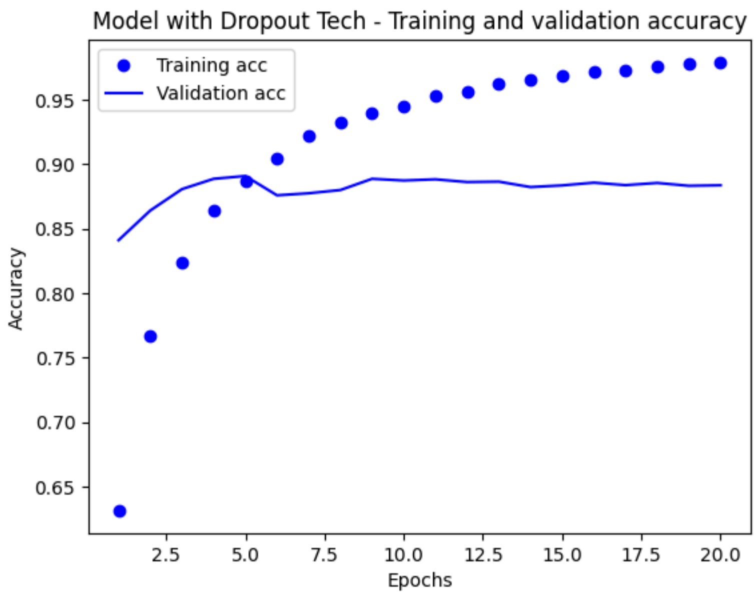
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Mod_Drp_Tech_dict = Mod_Drp_Tech.history
loss_values_Drp = Mod_Drp_Tech_dict["loss"]
val_loss_values_Drp = Mod_Drp_Tech_dict["val_loss"]
epochs = range(1, len(loss_values_Drp) + 1)
plt.plot(epochs, loss_values_Drp, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_Drp, "b", label="Validation loss")
plt.title("Model with Dropout Tech - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
acc_Drp = Mod_Drp_Tech_dict["accuracy"]
val_acc_Drp = Mod_Drp_Tech_dict["val_accuracy"]
plt.plot(epochs, acc_Drp, "bo", label="Training acc")
plt.plot(epochs, val_acc_Drp, "b", label="Validation acc")
plt.title("Model with Dropout Tech - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining

```
mod_drop = 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")
])
mod_drop.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
mod_drop.fit(x_train, y_train, epochs=9, batch_size=512) # Epochs selected 9 because it starts to stablize from 9
Mod_Drp_Tech_Results = mod_drop.evaluate(x_test, y_test)
```

```
Epoch 1/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.5973 - loss: 0.6517
Epoch 2/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.7918 - loss: 0.4832
Epoch 3/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.8535 - loss: 0.3797
Epoch 4/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.8856 - loss: 0.3182
Epoch 5/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9071 - loss: 0.2703
Epoch 6/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.9190 - loss: 0.2375
Epoch 7/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.9288 - loss: 0.2171
Epoch 8/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.9342 - loss: 0.1904
Epoch 9/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9426 - loss: 0.1814
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8825 - loss: 0.3326
```

```
Mod_Drp_Tech_Results
```

```
[0.3283398449420929, 0.8847600221633911]
```

```
mod_drop.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
array([[0.05492957],
       [0.9999943 ],
       [0.95317316],
       ...,
       [0.05558532],
       [0.03264352],
       [0.5672113 ]], dtype=float32)
```

Comparison of the Models

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

Mod_1_Hid_Lay_dict = Model_Hid_lay_1.history
Mod_1_Hid_Lay_dict.keys()

Model_3_Hid_Lay_dict = Model_3_Hid_Lay.history
Model_3_Hid_Lay_dict.keys()

Mod_32_Hid_Units_dict = Mod_32_Hid_Units.history
Mod_32_Hid_Units_dict.keys()

Mod_64_Hid_Units_dict = Mod_64_Hid_Units.history
Mod_64_Hid_Units_dict.keys()

Mod_MSE_LOSS_dict = Mod_MSE_LOSS.history
Mod_MSE_LOSS_dict.keys()

Mod_TANH_ACT_dict = Model_TANH_ACT.history
Mod_TANH_ACT_dict.keys()

Mod_Reg_Tech_dict = Mod_Reg_Tech.history
Mod_Reg_Tech_dict.keys()

Mod_Drp_Tech_dict = Mod_Drp_Tech.history
Mod_Drp_Tech_dict.keys()
```

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

Question 1 - Comparing Hidden layers with Base Model

```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": history,
    "Model_1_Hidden_Layer": Model_Hid_lay_1,
    "Model_3_Hidden_Layer": Model_3_Hid_Lay,
}

# Extract and display keys of histories
for model_name, model in model_histories.items():
    history_dict = model.history
    print(f"{model_name} history keys: {history_dict.keys()}")

# Function to plot training and validation accuracy/loss across models
def plot_metrics(metric):
    plt.figure(figsize=(10, 6))
    for model_name, model in model_histories.items():
        metric_values = model.history[metric]
        plt.plot(range(1, len(metric_values) + 1), metric_values, label=f"{model_name} ({metric})")

    plt.title(f'{metric.capitalize()} Comparison Across Models')
    plt.xlabel('Epochs')
    plt.ylabel(metric.capitalize())
    plt.legend()
    plt.show()

# Plot validation accuracy
plot_metrics('val_accuracy')

# Plot validation loss
plot_metrics('val_loss')

plot_metrics('accuracy')

plot_metrics('loss')
```

Show hidden output

Question 2 - Comparing Base model with Hidden Units value of 16, 32 and 64

```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": history,
    "Model_32_Hidden_Units": Mod_32_Hid_Units,
    "Model_64_Hidden_Units": Mod_64_Hid_Units,
}

# Extract and display keys of histories
for model_name, model in model_histories.items():
    history_dict = model.history
    print(f"{model_name} history keys: {history_dict.keys()}")

# Function to plot training and validation accuracy/loss across models
def plot_metrics(metric):
    plt.figure(figsize=(10, 6))
    for model_name, model in model_histories.items():
        metric_values = model.history[metric]
        plt.plot(range(1, len(metric_values) + 1), metric_values, label=f"{model_name} ({metric})")

    plt.title(f'{metric.capitalize()} Comparison Across Models')
    plt.xlabel('Epochs')
    plt.ylabel(metric.capitalize())
    plt.legend()
    plt.show()

# Plot validation accuracy
plot_metrics('val_accuracy')

# Plot validation loss
plot_metrics('val_loss')

plot_metrics('accuracy')

plot_metrics('loss')
```

Show hidden output

Question 3 - Comparing of MSE loss function

```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": history,
    "Model_MSE_Loss": Mod_MSE_LOSS,
 }

# Extract and display keys of histories
for model_name, model in model_histories.items():
    history_dict = model.history
    print(f"{model_name} history keys: {history_dict.keys()}")

# Function to plot training and validation accuracy/loss across models
def plot_metrics(metric):
    plt.figure(figsize=(10, 6))
    for model_name, model in model_histories.items():
        metric_values = model.history[metric]
        plt.plot(range(1, len(metric_values) + 1), metric_values, label=f"{model_name} ({metric})")

    plt.title(f'{metric.capitalize()} Comparison Across Models')
    plt.xlabel('Epochs')
    plt.ylabel(metric.capitalize())
    plt.legend()
    plt.show()

# Plot validation accuracy
plot_metrics('val_accuracy')

# Plot validation loss
plot_metrics('val_loss')

plot_metrics('accuracy')
```

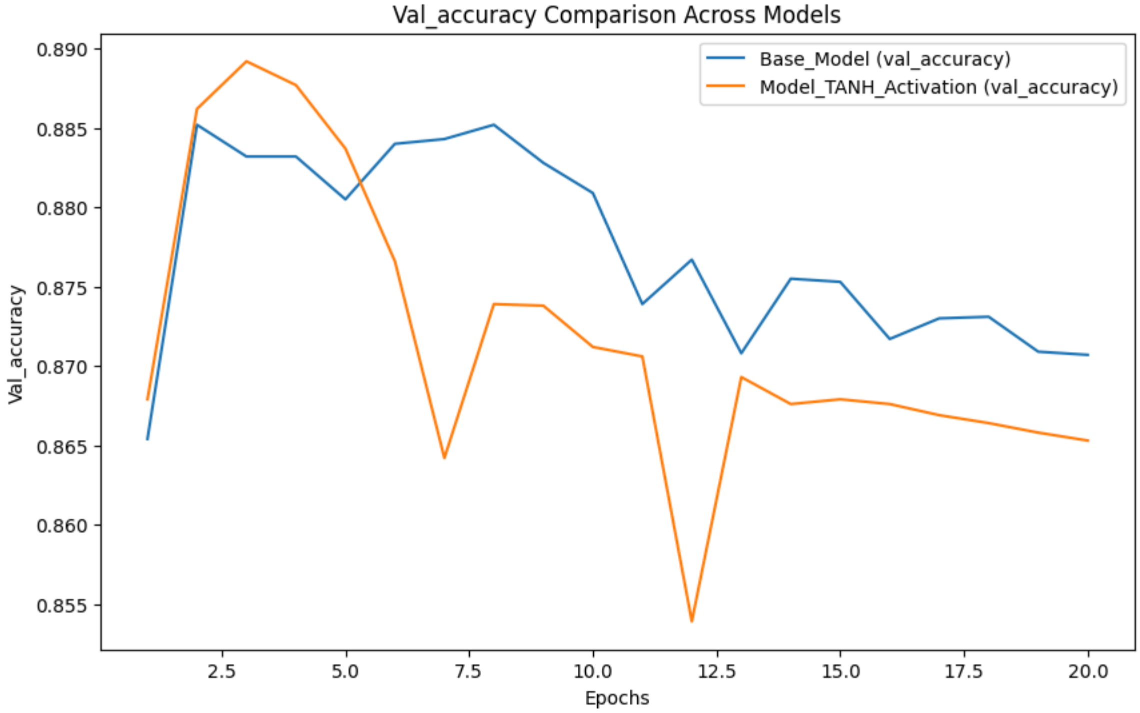
```

plot_metrics('loss')
```

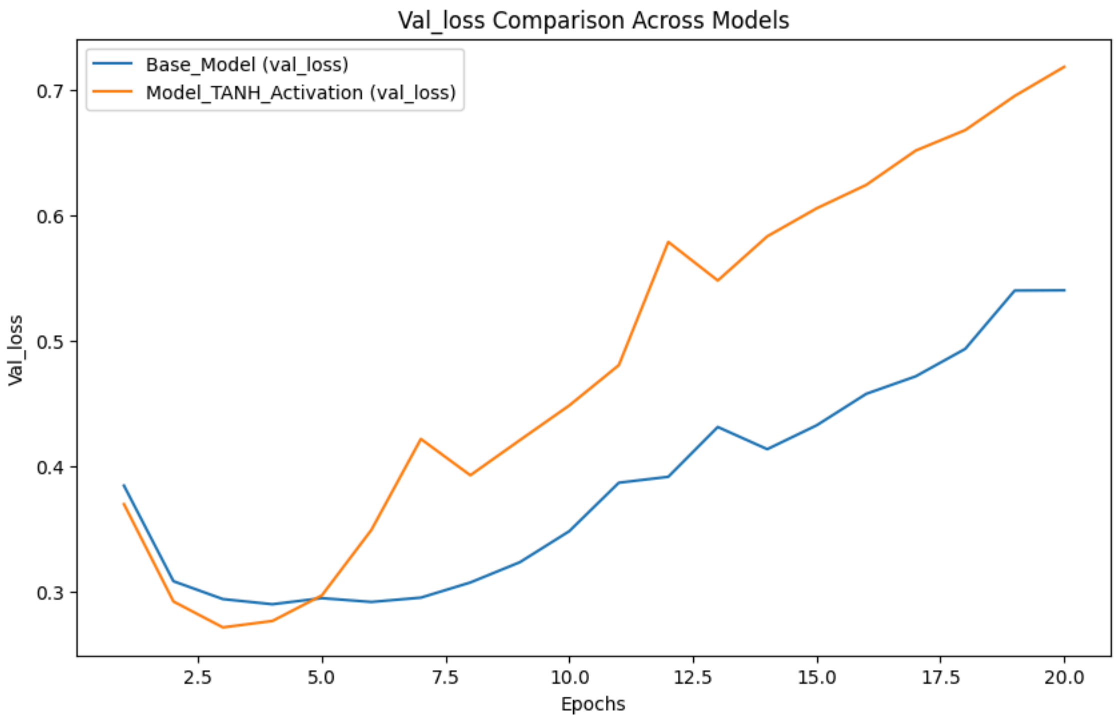
Show hidden output

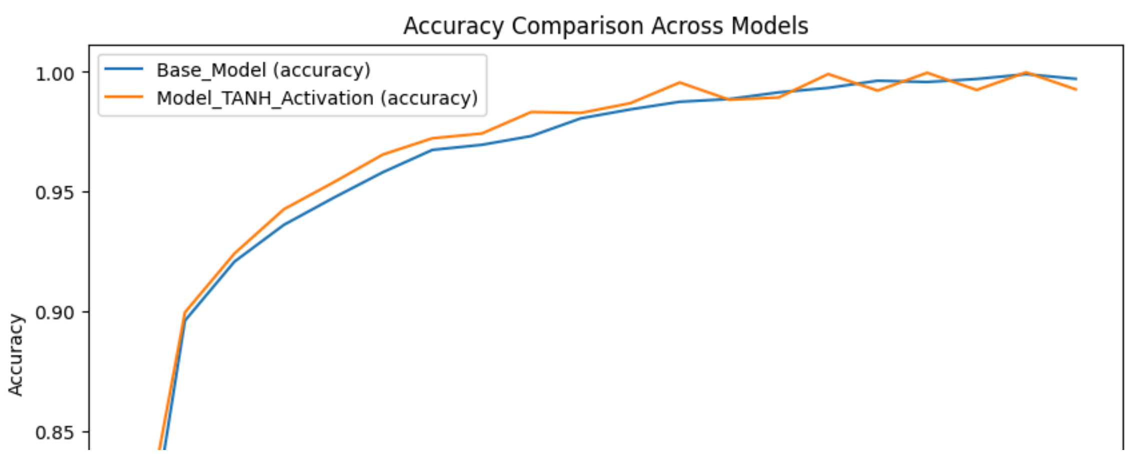
Question 4 - Comparing of Tanh activation with base model

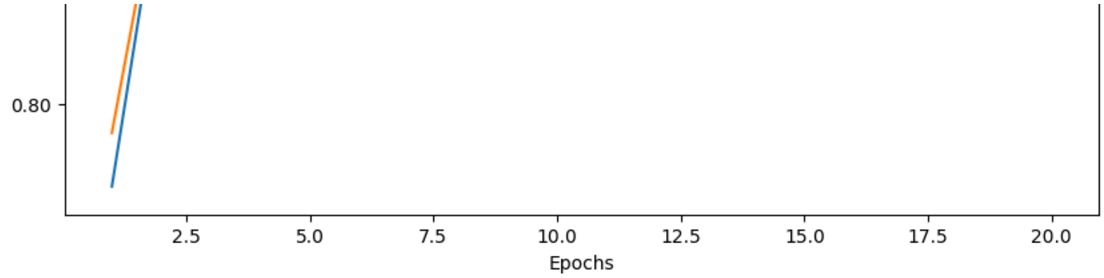
```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": history,
    "Model_TANH_Activation": Model_TANH_ACT,
}

# Extract and display keys of histories
for model_name, model in model_histories.items():
    history_dict = model.history
    print(f"{model_name} history keys: {history_dict.keys()}")

# Function to plot training and validation accuracy/loss across models
def plot_metrics(metric):
    plt.figure(figsize=(10, 6))
    for model_name, model in model_histories.items():
        metric_values = model.history[metric]
        plt.plot(range(1, len(metric_values) + 1), metric_values, label=f"{model_name} ({metric})")

    plt.title(f'{metric.capitalize()} Comparison Across Models')
    plt.xlabel('Epochs')
    plt.ylabel(metric.capitalize())
    plt.legend()
    plt.show()

# Plot validation accuracy
plot_metrics('val_accuracy')

# Plot validation loss
plot_metrics('val_loss')

plot_metrics('accuracy')

plot_metrics('loss')
```

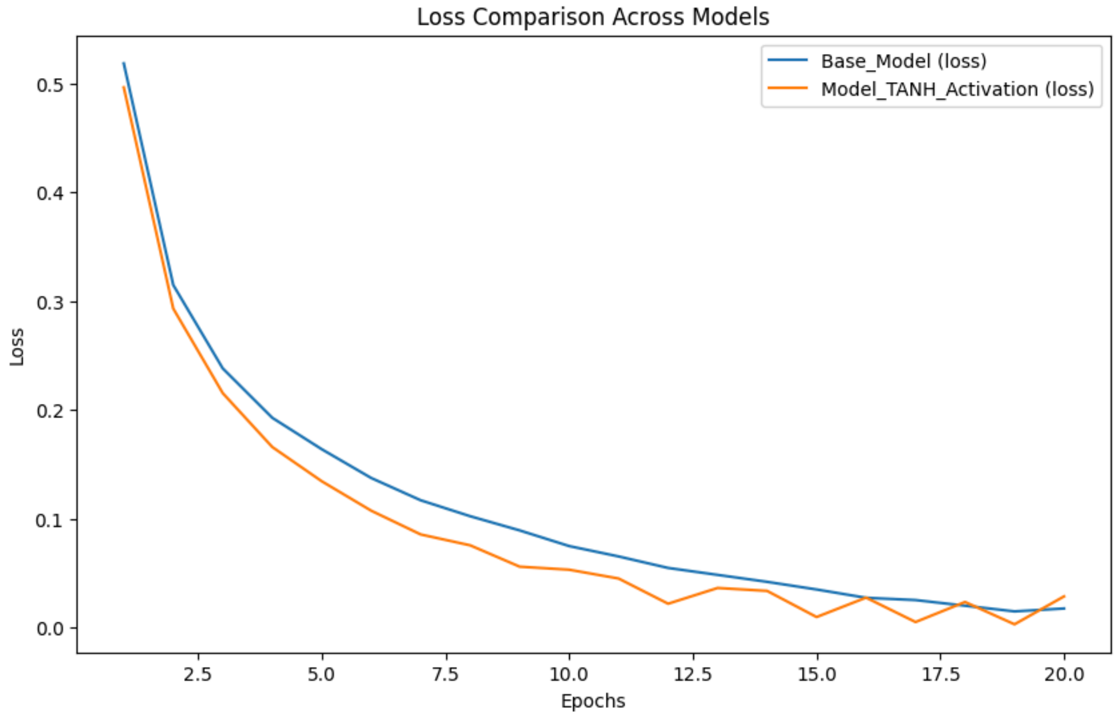
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_TANH_Activation history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







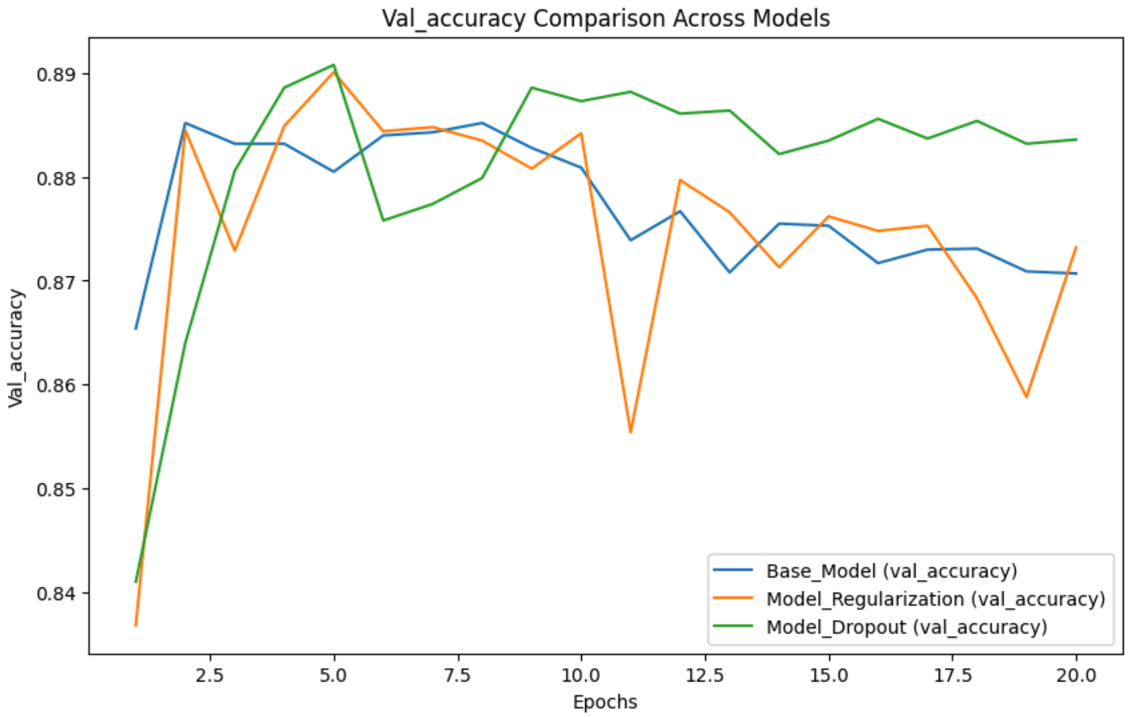

0.80
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs


Loss Comparison Across Models
Base_Model (loss)
Model_TANH_Activation (loss)
Loss
0.0
0.1
0.2
0.3
0.4
0.5
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs

Question 5 - Comparison of L2 regularization, Dropout and Base model

```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": history,
    "Model_Regularization": Mod_Reg_Tech,
    "Model_Dropout": Mod_Drp_Tech
}

# Extract and display keys of histories
for model_name, model in model_histories.items():
    history_dict = model.history
    print(f"{model_name} history keys: {history_dict.keys()}")

# Function to plot training and validation accuracy/loss across models
def plot_metrics(metric):
    plt.figure(figsize=(10, 6))
    for model_name, model in model_histories.items():
        metric_values = model.history[metric]
        plt.plot(range(1, len(metric_values) + 1), metric_values, label=f"{model_name} ({metric})")

    plt.title(f'{metric.capitalize()} Comparison Across Models')
    plt.xlabel('Epochs')
    plt.ylabel(metric.capitalize())
    plt.legend()
    plt.show()

# Plot validation accuracy
plot_metrics('val_accuracy')

# Plot validation loss
plot_metrics('val_loss')

plot_metrics('accuracy')

plot_metrics('loss')
```

```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_Regularization history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_Dropout history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```



Accuracy Comparison Across Models