AML – 64016 – Assignment 2

apeter@kent.edu

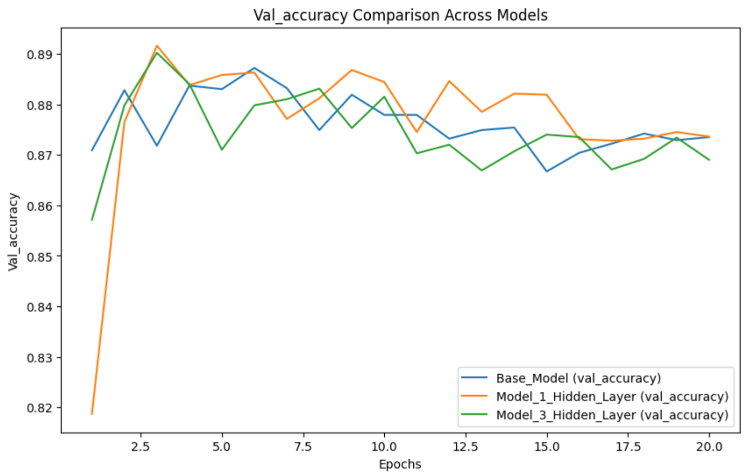
GitHub link - https://github.com/Aloysius95/apeter_64061/tree/4546a7de2a5e2b7b527a4f7a0e20fc0765add2ae/Assignment2

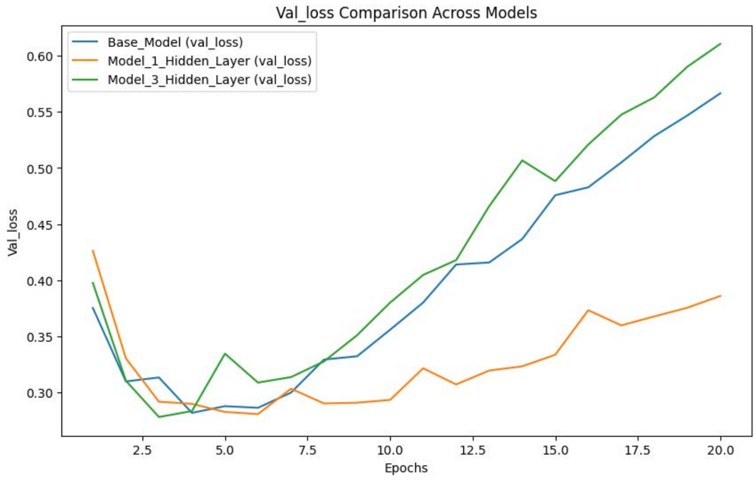
1. **You used two hidden layers. Try using one or three hidden layers and see how doing so affects validation and test accuracy.**

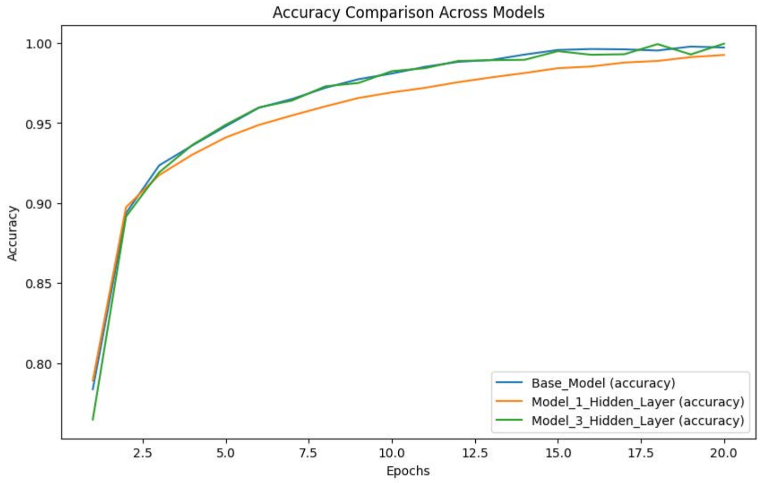
- The base model with 2 hidden layers shows the highest validation accuracy and the lowest loss. Which outperforms both the model with 1 and 3 hidden layers.
- Model 1 with 1 hidden layer performs slightly worse than the base model in both validation and training accuracy.
- Model 3 with 3 hidden layer does not improve the performance, which shows that the additional layers do not always enhance the performance and also results in instability.
- Overall, the base model with two hidden layers provides the most stable and optimal performance.

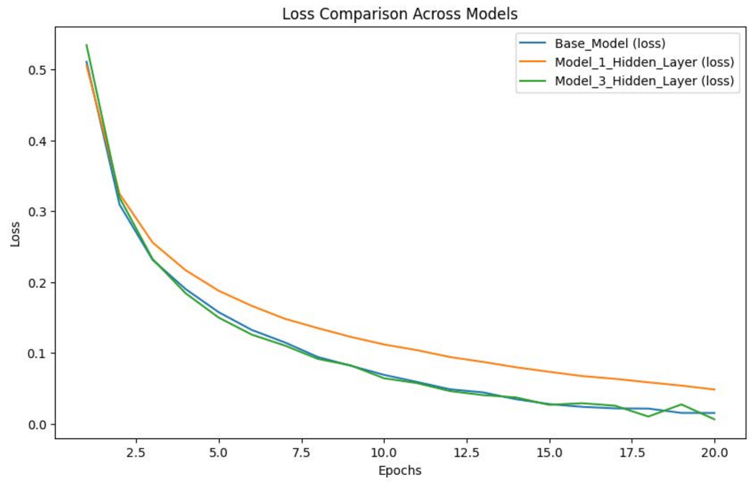
	Test loss	Test Accuracy
Base Model	0.28	0.89
1HL	0.28	0.89
3HL	0.37	0.87

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







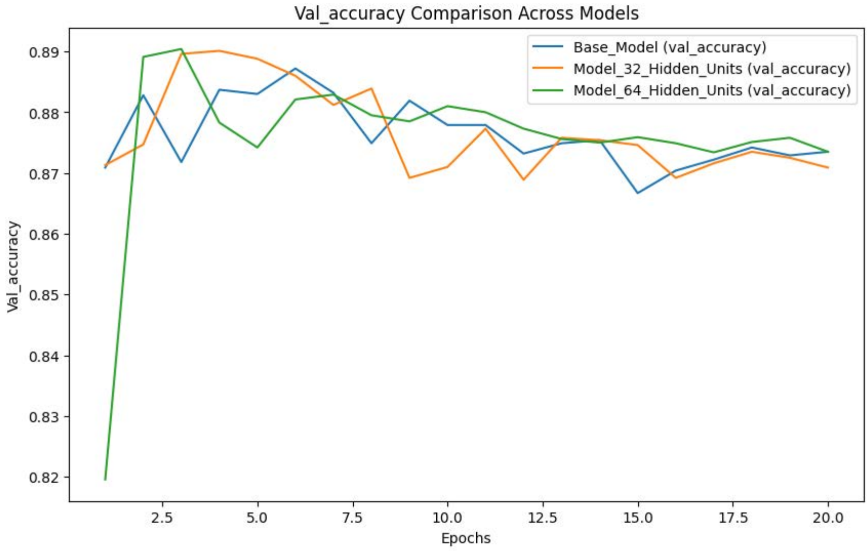


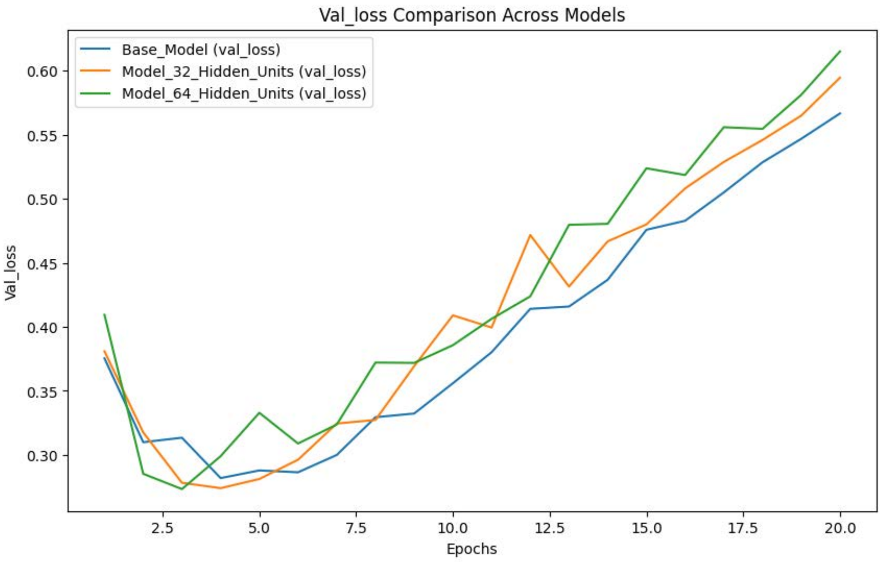
2. **Try using layers with more hidden units or fewer hidden units: 32 units, 64 units, and so on.**

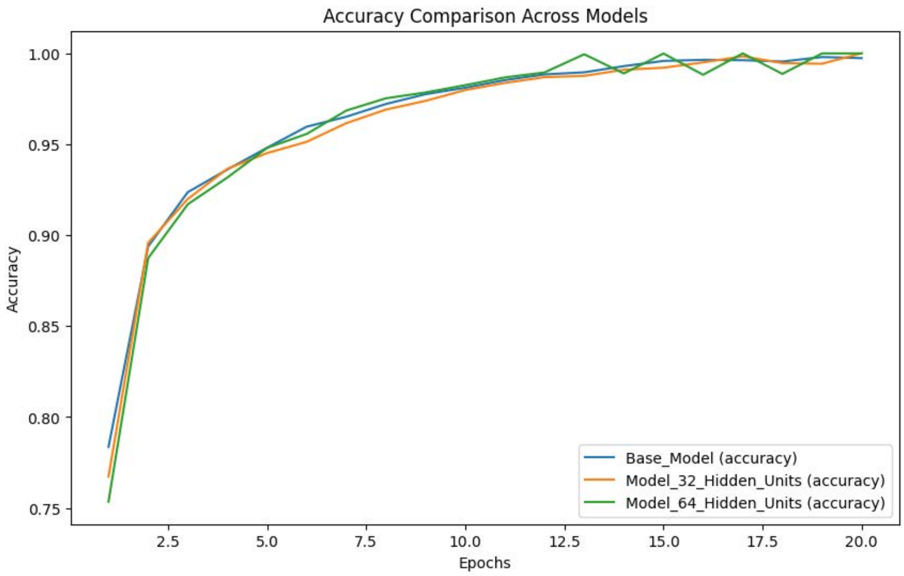
- In validation accuracy the model with 32 units actually performs better than 16 and 64 units. The model with 64 units shows lower and more variable validation accuracy while the model with 32 units performed similar to the base model which is 16 units.

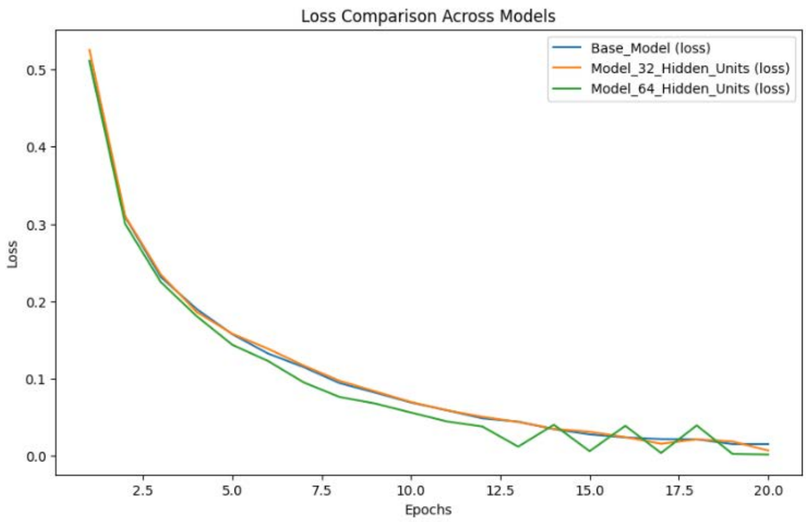
- The model with 64 hidden units has the lowest validation loss when compared to 32 and 16 hidden units model, especially at higher epochs which suggests that increasing the hidden units may lead to overfitting and suboptimal generalization.
- All the models with units 13, 32 and 64 performed similar to each other in training accuracy and loss but the base model performed slightly better as a whole.
- To conclude increasing the no of hidden units from 16 to 32 to 64 did not improve the model that much in these models but it gave a lot of volatility because of which the base model is a better option among them.

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





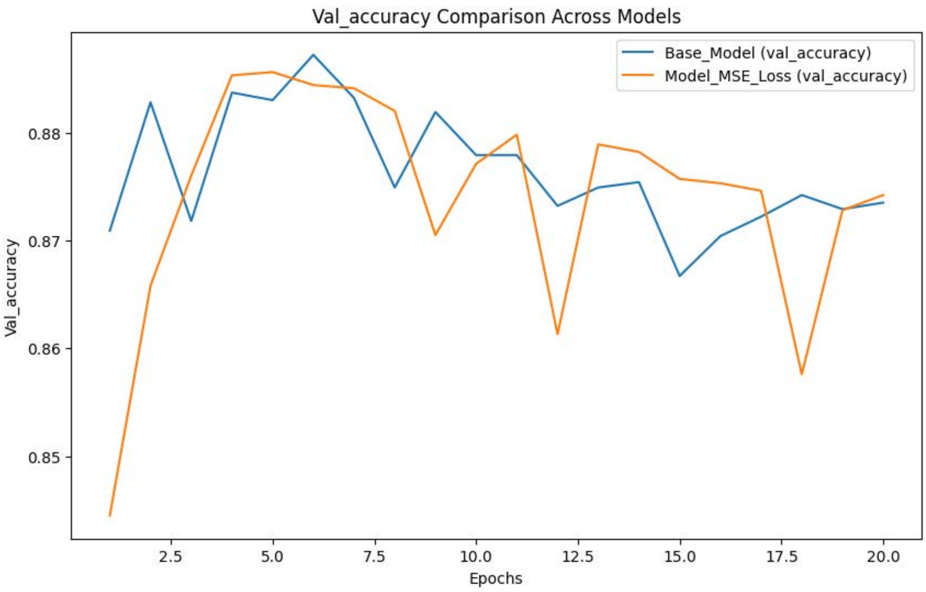


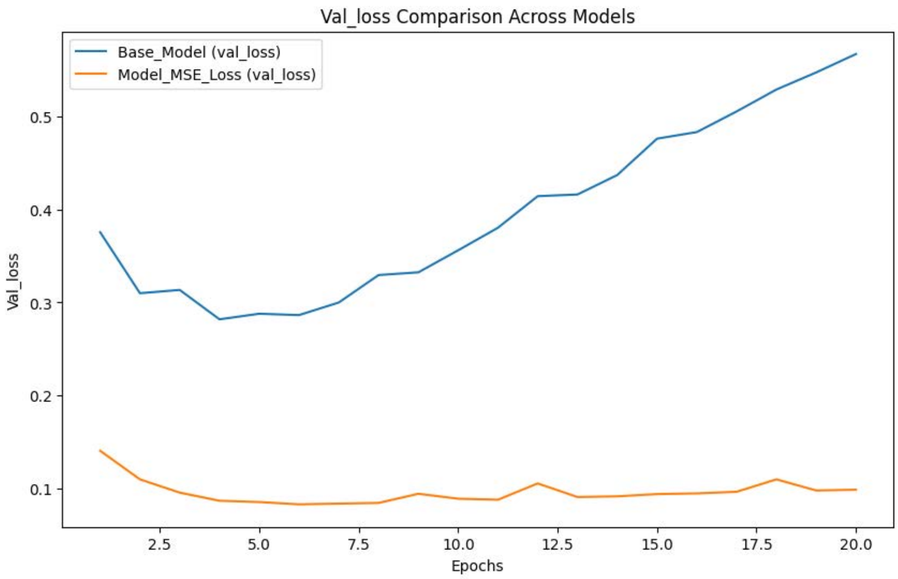


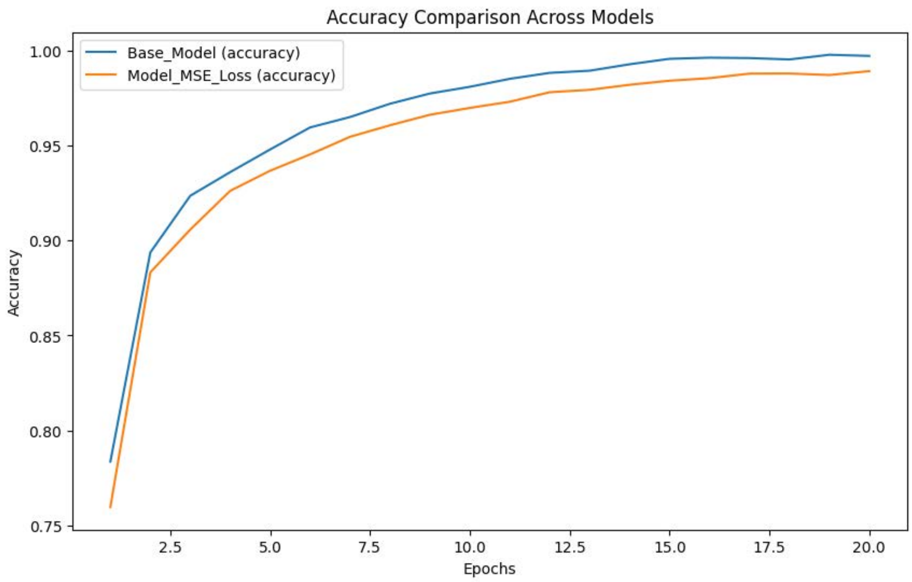
3. Try using the MSE loss function instead of binary cross entropy.

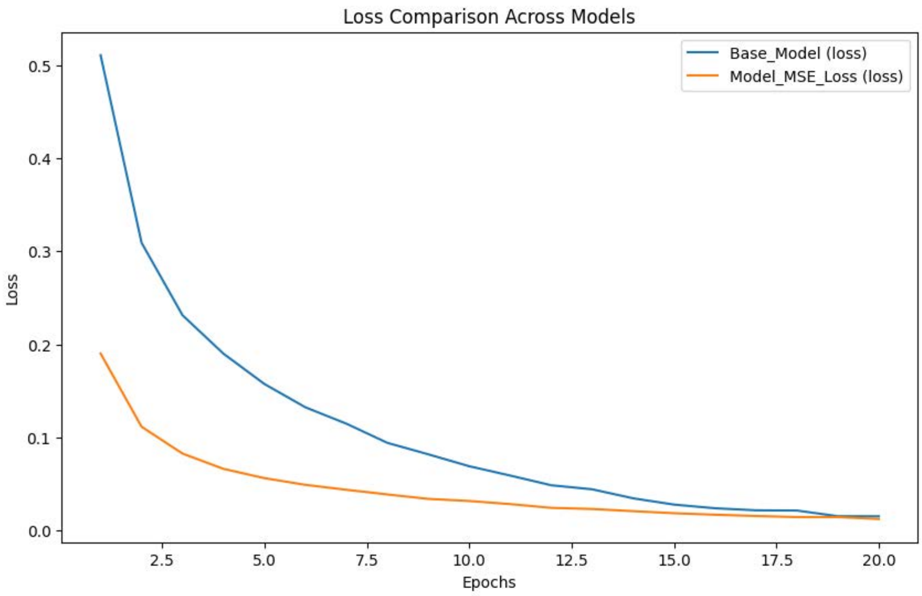
- BCE is better suited for binary classification problem, in this case the MSE performs significantly better than BCE.
- The MSE majorly outperforms in validation loss comparison, where it shows a lower val loss compared to the base model. This shows the MSE model is better able to minimize the error between predicted and true values.
- The MSE model gives better results with less training (with lesser EPOCHS)
- In conclusion the MSE model is clearly better than using BCE model for this specific model building.

	Test loss	**Test Accuracy**
Base Model	0.28	0.89
MSE	0.09	0.88


Val_accuracy Comparison Across Models
Base_Model (val_accuracy)
Model_MSE_Loss (val_accuracy)
Val_accuracy
Epochs
0.85
0.86
0.87
0.88
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_MSE_Loss (val_loss)
Val_loss
Epochs
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

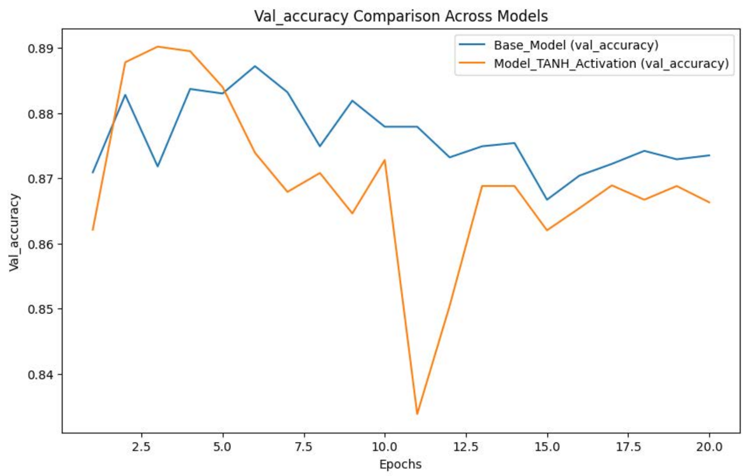

Accuracy Comparison Across Models
Base_Model (accuracy)
Model_MSE_Loss (accuracy)
Accuracy
Epochs
0.75
0.80
0.85
0.90
0.95
1.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

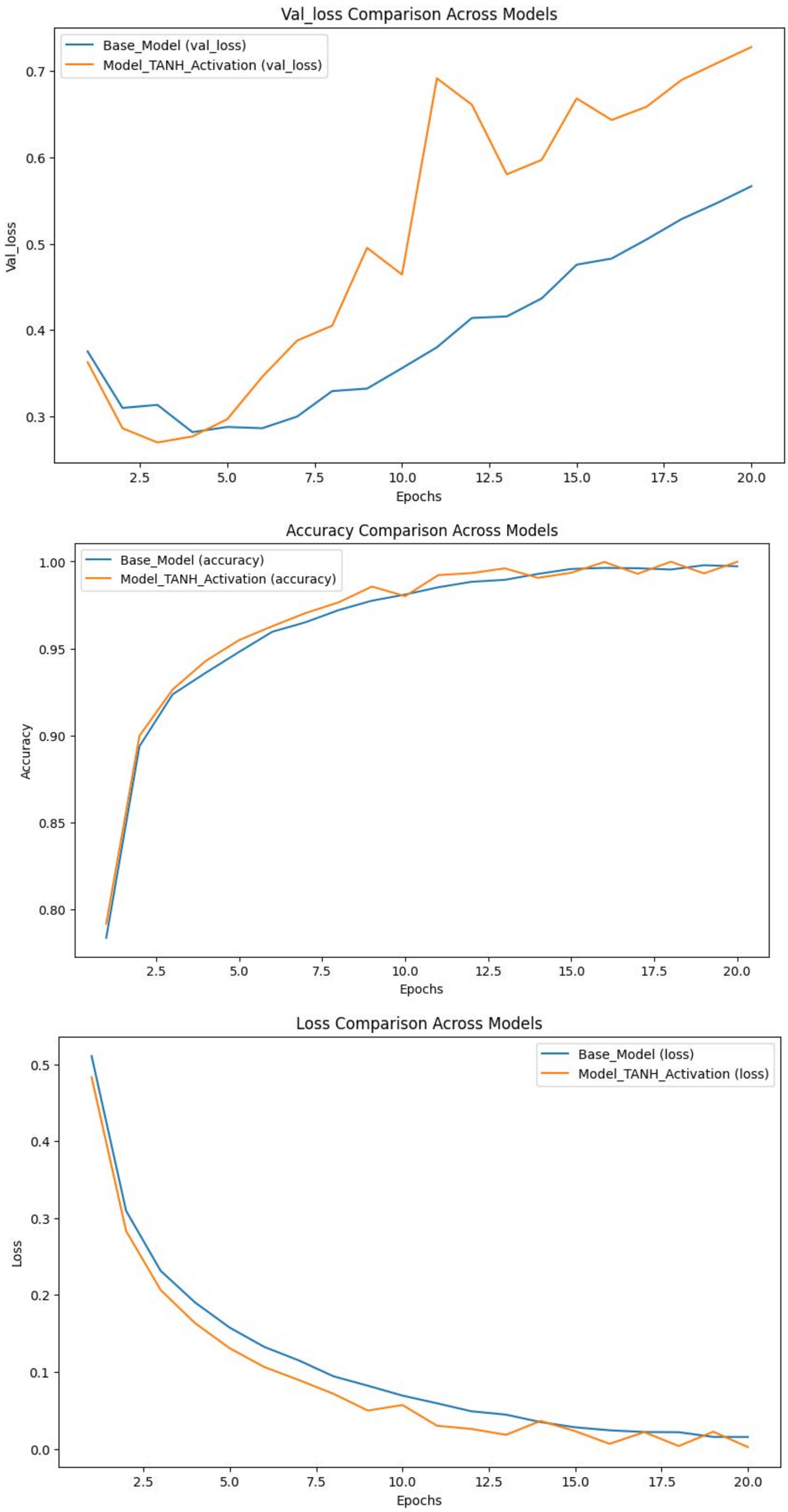


4. **Try using the tanh activation (an activation that was popular in the early days of neural networks) instead of relu**

- The relu activation (Base line model) performs better than tanh activation, where relu's accuracy is better.
- Relu activation model shows a lower loss compared to the tanh model which shows that the relu model is better to minimize the error between predicted and true value.
- In conclusion, relu is a better option because it outperforms tanh in accuracy and has a lower loss.

	Test loss	**Test Accuracy**
Base Model	0.28	0.89
Tanh	0.28	0.88

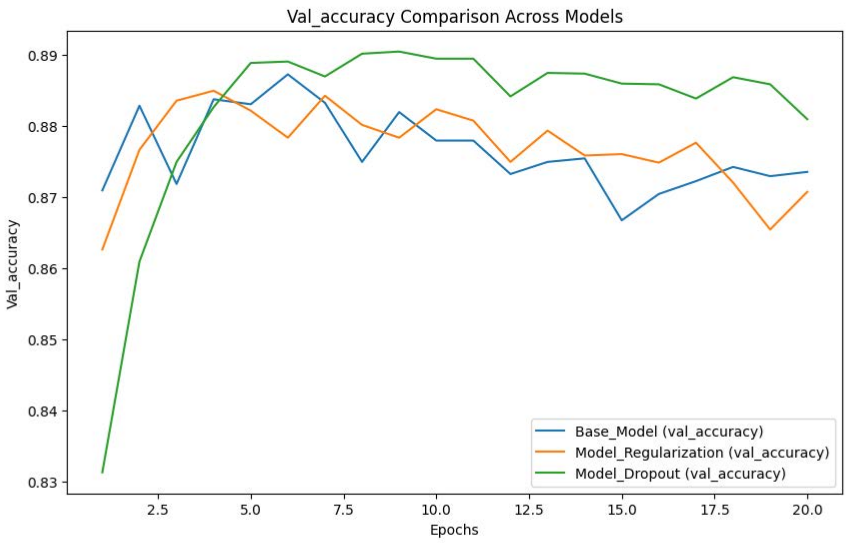


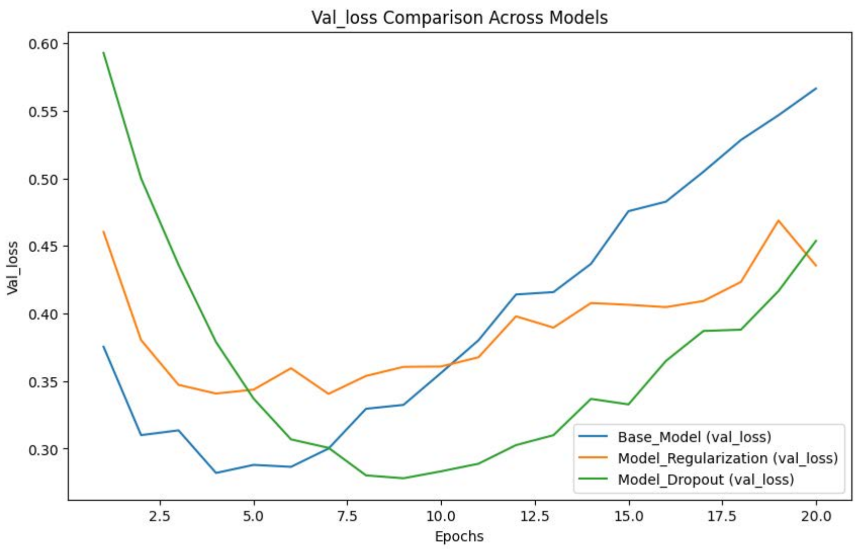

Val_loss Comparison Across Models
Base_Model (val_loss)
Model_TANH_Activation (val_loss)
Val_loss
0.3
0.4
0.5
0.6
0.7
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs
Accuracy Comparison Across Models
Base_Model (accuracy)
Model_TANH_Activation (accuracy)
Accuracy
0.80
0.85
0.90
0.95
1.00
2.5
5.0
7.5
10.0
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

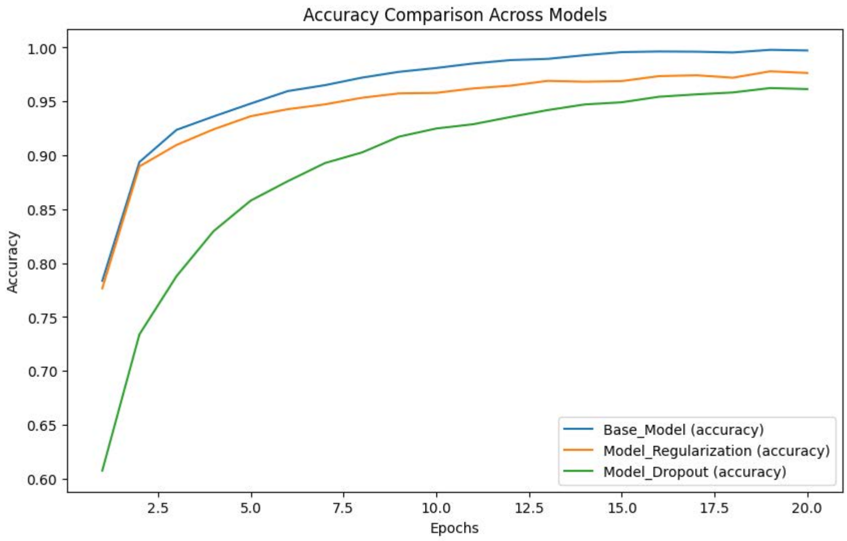
5. **Use any technique we studied in class, and these include regularization, dropout, etc., to get your model to perform better on validation.**

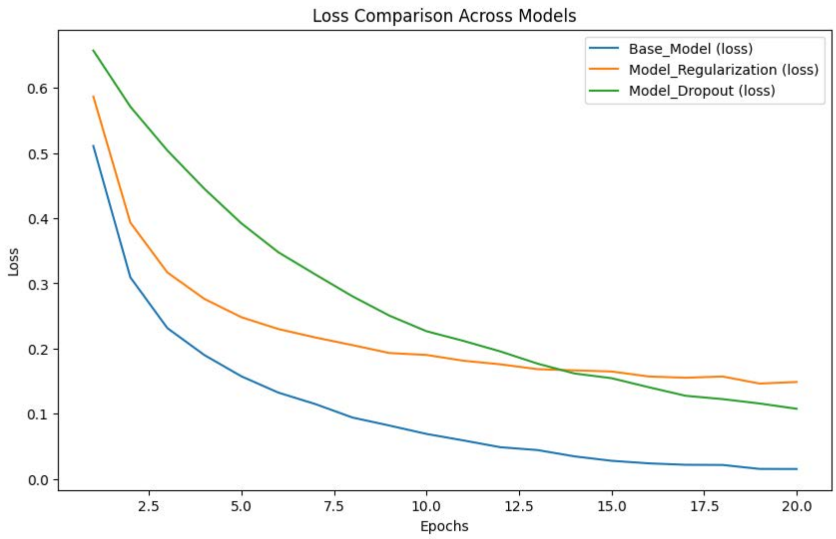
- Dropout optimization has the best accuracy compared to the base line and L2 regularization.
- Dropout again performs better at the loss where it has the lowest error rate, with L2 close behind and the base model being not that effective.
- Mostly compared to all the graphs the dropout is a better optimization method for this model because it performs the best compared to baseline and L2.

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







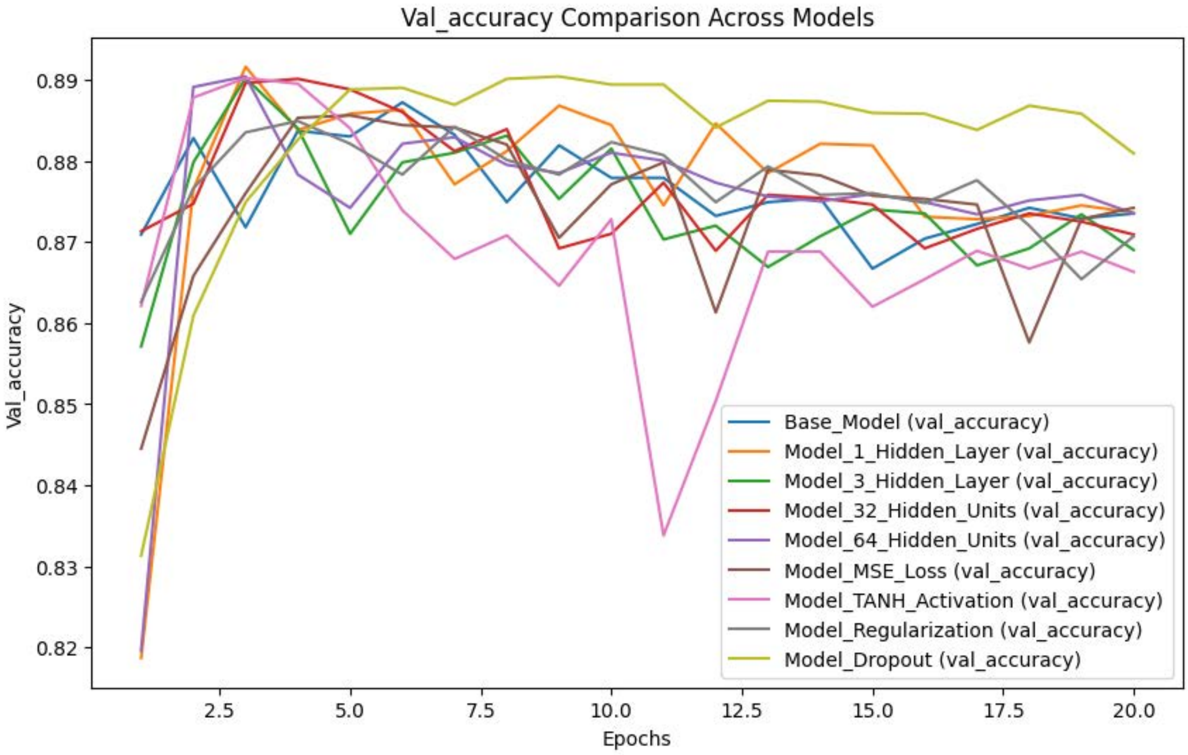


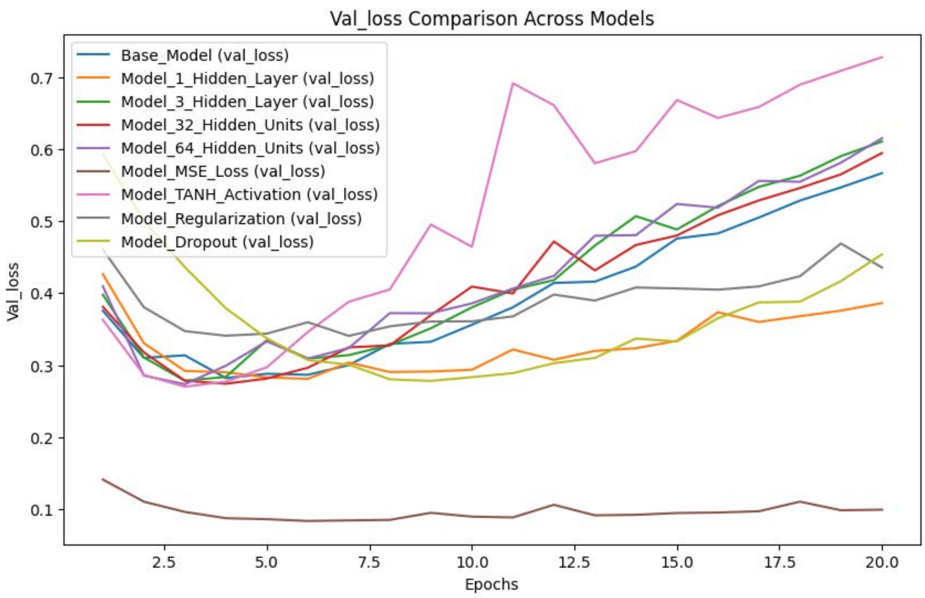
Comparison of all the models.

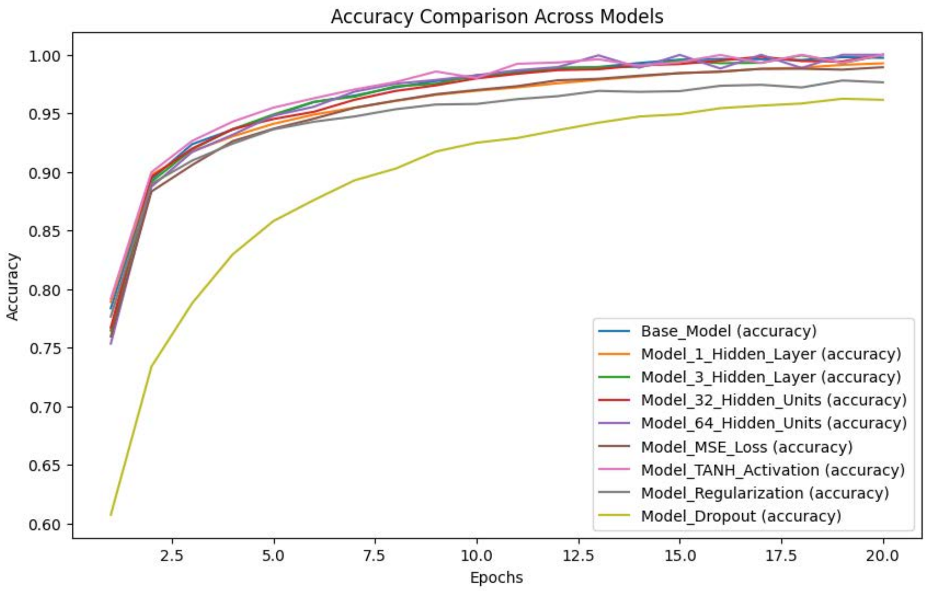
- For the validation accuracy model among all the models, the 32 hidden units performed the best.
- For the accuracy comparison, tanh and the base model together performed very similar, and they were the best in this comparison.
- For the validation loss and the loss comparison the 64 hidden units model came out to be the best when compared to other models.
- To conclude the model a good balance and performed equally acceptable on all fronts was the 64 hidden units.

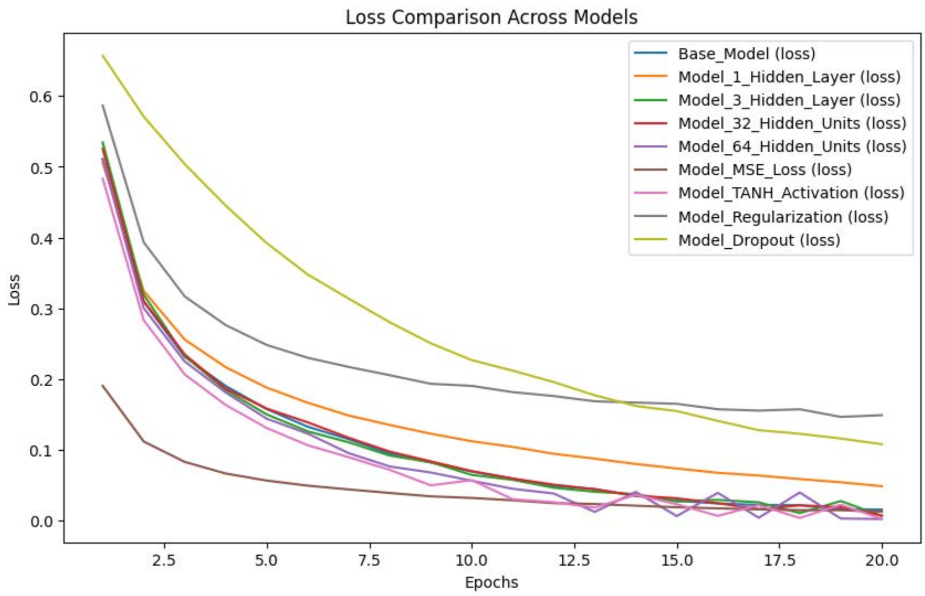
	Test loss	**Test Accuracy**
Base Model	0.28	0.89
1HL	0.28	0.89

3HL	0.37	0.87
32HU	0.29	0.88
64HU	0.29	0.88
MSE	0.09	0.88
Tanh	0.28	0.88
L2	0.34	0.88
Dropout	0.32	0.88









Following is the code explanation

- Inputting the dataset and preparing the data – the code was given by professor.
- Each model was created differently for model building to make it easier to compare them, in total there were around 9 models.
- Then each model was trained and plotted to see its accuracy and loss.
- Each model was retrained as per the base model given by the professor.
- Based on the questions each model were put together to compare and all the models were compared together, the results are above.

** *Because of the length of the model the google colab page was printed and attached at the end of the report.*

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]
```

Show hidden output

```
train_labels[0]
```

```
1
```

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

```
9999
```

Decoding reviews 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 the model different configurations

0. Model given by professor - Base point (model)

```
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. (Question 1) Building the model with 1 hidden layer (model_1_HL)

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

model_1_HL = keras.Sequential([
    layers.Dense(16, activation="relu"), # Building the model with 1 hidden layer
    layers.Dense(1, activation="sigmoid")
])

model_1_HL.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

2. (Question 1) Building the model with 3 hidden layer (model_3_hl)

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

model_3_HL = 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")
])

model_3_HL.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

3. (Question 2) Building the model with fewer hidden units 32 (model_32_HU)

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

model_32_HU = keras.Sequential([
    layers.Dense(32, activation="relu"), # hidden units 32
    layers.Dense(32, activation="relu"), # hidden units 32
    layers.Dense(1, activation="sigmoid")
    ])

model_32_HU.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

4. (Question 2) Building the model with higher hidden units 64 (model64_HU)

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

model_64_HU = keras.Sequential([
    layers.Dense(64, activation="relu"), # hidden units 64
    layers.Dense(64, activation="relu"), # hidden units 64
    layers.Dense(1, activation="sigmoid")
    ])

model_64_HU.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

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

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

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

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

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

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

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

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

```
from tensorflow.keras import regularizers

model_reg = keras.Sequential([
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)), # Applied L2 regularization (0.001 - common acceptec
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)), # Applied L2 regularization (0.001 - common acceptec
    layers.Dense(1, activation="sigmoid")
])

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

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

```
model_drp = 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")
])

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

Creating a validation set

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

Training your model

0. Base model

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

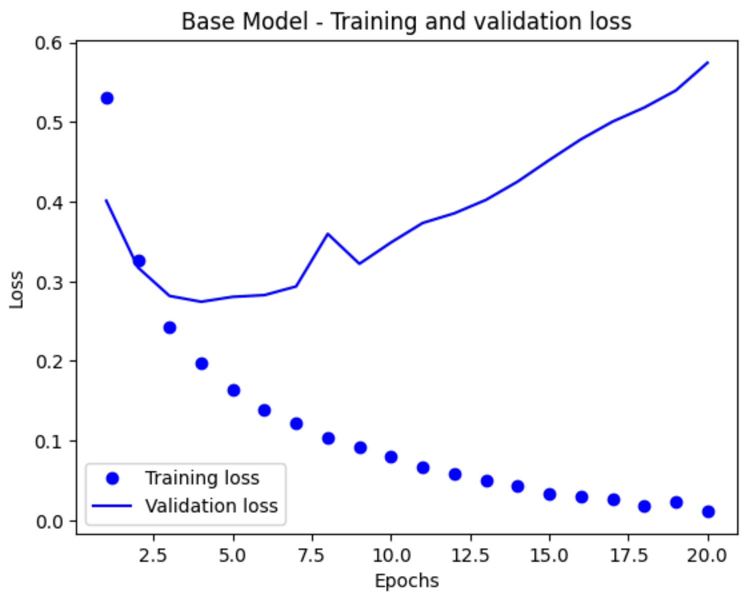
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 64ms/step - accuracy: 0.6956 - loss: 0.6023 - val_accuracy: 0.8656 - val_loss: 0.4013
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.8949 - loss: 0.3472 - val_accuracy: 0.8802 - val_loss: 0.3176
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9216 - loss: 0.2514 - val_accuracy: 0.8911 - val_loss: 0.2818
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9353 - loss: 0.1987 - val_accuracy: 0.8897 - val_loss: 0.2744
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.9477 - loss: 0.1634 - val_accuracy: 0.8854 - val_loss: 0.2807
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9574 - loss: 0.1372 - val_accuracy: 0.8871 - val_loss: 0.2828
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9652 - loss: 0.1178 - val_accuracy: 0.8852 - val_loss: 0.2937
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9738 - loss: 0.0975 - val_accuracy: 0.8703 - val_loss: 0.3597
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9722 - loss: 0.0936 - val_accuracy: 0.8831 - val_loss: 0.3221
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9822 - loss: 0.0717 - val_accuracy: 0.8754 - val_loss: 0.3489
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9846 - loss: 0.0653 - val_accuracy: 0.8704 - val_loss: 0.3734
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9886 - loss: 0.0553 - val_accuracy: 0.8723 - val_loss: 0.3855
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.9905 - loss: 0.0481 - val_accuracy: 0.8786 - val_loss: 0.4024
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 54ms/step - accuracy: 0.9939 - loss: 0.0376 - val_accuracy: 0.8771 - val_loss: 0.4253
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9956 - loss: 0.0315 - val_accuracy: 0.8754 - val_loss: 0.4524
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9949 - loss: 0.0302 - val_accuracy: 0.8674 - val_loss: 0.4784
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9978 - loss: 0.0226 - val_accuracy: 0.8731 - val_loss: 0.5007
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9985 - loss: 0.0189 - val_accuracy: 0.8718 - val_loss: 0.5183
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9962 - loss: 0.0219 - val_accuracy: 0.8725 - val_loss: 0.5398
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9996 - loss: 0.0113 - val_accuracy: 0.8697 - val_loss: 0.5746
```

```
Base_model_dict = Base_model.history
Base_model_dict.keys()
```

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

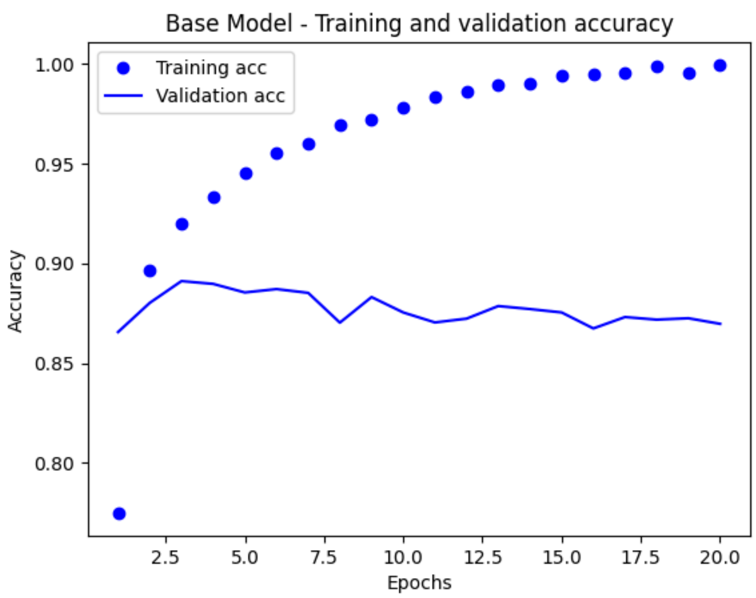
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Base_model_dict = Base_model.history
loss_values_0 = Base_model_dict["loss"]
val_loss_values_0 = Base_model_dict["val_loss"]
epochs = range(1, len(loss_values_0) + 1)
plt.plot(epochs, loss_values_0, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_0, "b", label="Validation loss")
plt.title("Base Model - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
acc_0 = Base_model_dict["accuracy"]
val_acc_0 = Base_model_dict["val_accuracy"]
plt.plot(epochs, acc_0, "bo", label="Training acc")
plt.plot(epochs, val_acc_0, "b", label="Validation acc")
plt.title("Base Model - Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining

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

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 38ms/step - accuracy: 0.7307 - loss: 0.5955
Epoch 2/4
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.8918 - loss: 0.3226
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9185 - loss: 0.2375
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9257 - loss: 0.2012
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8875 - loss: 0.2796
```

```
Base_model_results
```

```
[0.2787157893180847, 0.8880000114440918]
```

Using Trained data to predict

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

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.21400136],
       [0.9994859 ],
       [0.81648225],
       ...,
       [0.07354003],
       [0.08245087],
       [0.5075143 ]], dtype=float32)
```

1. Model With 1 Hidden Layer

```
Model_1_Hidden_Layer = model_1_HL.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

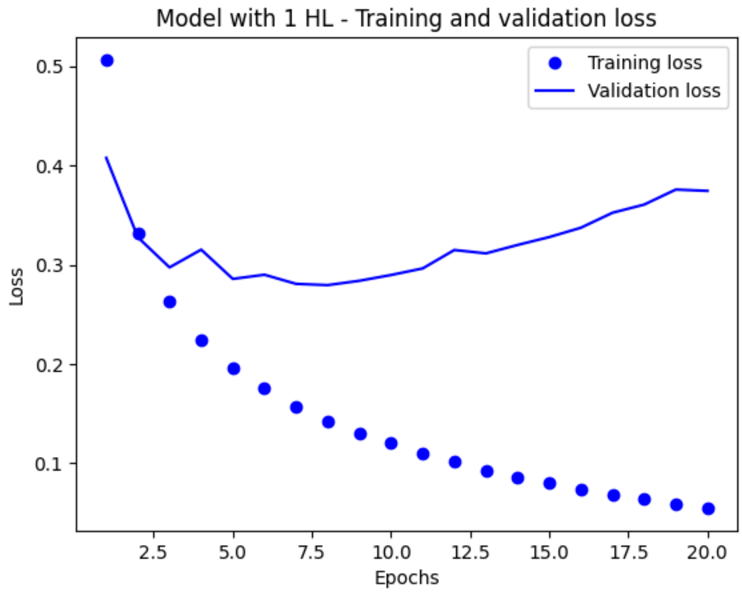
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 75ms/step - accuracy: 0.7208 - loss: 0.5762 - val_accuracy: 0.8574 - val_loss: 0.4077
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8909 - loss: 0.3488 - val_accuracy: 0.8792 - val_loss: 0.3276
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9125 - loss: 0.2716 - val_accuracy: 0.8854 - val_loss: 0.2974
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9273 - loss: 0.2286 - val_accuracy: 0.8721 - val_loss: 0.3153
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9359 - loss: 0.2011 - val_accuracy: 0.8861 - val_loss: 0.2858
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9421 - loss: 0.1779 - val_accuracy: 0.8820 - val_loss: 0.2900
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9560 - loss: 0.1547 - val_accuracy: 0.8864 - val_loss: 0.2808
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9557 - loss: 0.1444 - val_accuracy: 0.8863 - val_loss: 0.2795
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9620 - loss: 0.1299 - val_accuracy: 0.8851 - val_loss: 0.2839
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9665 - loss: 0.1200 - val_accuracy: 0.8849 - val_loss: 0.2897
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.9744 - loss: 0.1056 - val_accuracy: 0.8836 - val_loss: 0.2963
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9773 - loss: 0.0964 - val_accuracy: 0.8780 - val_loss: 0.3149
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9773 - loss: 0.0909 - val_accuracy: 0.8816 - val_loss: 0.3115
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9780 - loss: 0.0866 - val_accuracy: 0.8814 - val_loss: 0.3200
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9854 - loss: 0.0738 - val_accuracy: 0.8800 - val_loss: 0.3279
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9850 - loss: 0.0700 - val_accuracy: 0.8802 - val_loss: 0.3373
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9867 - loss: 0.0687 - val_accuracy: 0.8737 - val_loss: 0.3524
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9875 - loss: 0.0638 - val_accuracy: 0.8783 - val_loss: 0.3606
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9892 - loss: 0.0570 - val_accuracy: 0.8767 - val_loss: 0.3758
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.9911 - loss: 0.0533 - val_accuracy: 0.8763 - val_loss: 0.3744
```

```
Model_1_Hidden_Layer_dict = Model_1_Hidden_Layer.history
Model_1_Hidden_Layer_dict.keys()
```

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

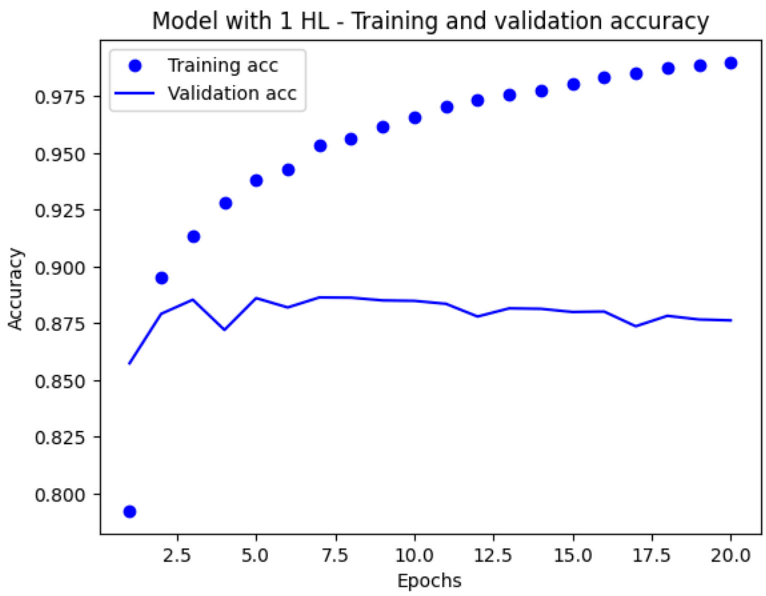
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_1_Hidden_Layer_dict = Model_1_Hidden_Layer.history
loss_values_1 = Model_1_Hidden_Layer_dict["loss"]
val_loss_values_1 = Model_1_Hidden_Layer_dict["val_loss"]
epochs = range(1, len(loss_values_1) + 1)
plt.plot(epochs, loss_values_1, "bo", label="Training loss")
plt.plot(epochs, val_loss_values_1, "b", label="Validation loss")
plt.title("Model with 1 HL - Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
acc_1 = Model_1_Hidden_Layer_dict["accuracy"]
val_acc_1 = Model_1_Hidden_Layer_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

```
model_1_HL = keras.Sequential([
    layers.Dense(16, activation="relu"), # 1 Hidden Layer
    layers.Dense(1, activation="sigmoid")
])
model_1_HL.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model_1_HL.fit(x_train, y_train, epochs=4, batch_size=512)
Model_1_Hidden_Layer_Results = model_1_HL.evaluate(x_test, y_test)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.7511 - loss: 0.5470
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8972 - loss: 0.3076
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9164 - loss: 0.2411
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9276 - loss: 0.2111
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8820 - loss: 0.2882
```

```
Model_1_Hidden_Layer_Results
```

```
[0.2891532778739929, 0.8822000026702881]
```

Using Trained data to predict

```
model_1_HL.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.21235158],
       [0.999259  ],
       [0.6847338 ],
       ...,
       [0.10059176],
       [0.09167492],
       [0.3551244 ]], dtype=float32)
```

2. Model With 3 Hidden Layer

```
Model_3_Hidden_Layer = model_3_HL.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - accuracy: 0.6750 - loss: 0.6091 - val_accuracy: 0.8727 - val_loss: 0.3798
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.8946 - loss: 0.3254 - val_accuracy: 0.8691 - val_loss: 0.3254
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9244 - loss: 0.2301 - val_accuracy: 0.8886 - val_loss: 0.2784
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9449 - loss: 0.1734 - val_accuracy: 0.8874 - val_loss: 0.2830
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9540 - loss: 0.1428 - val_accuracy: 0.8867 - val_loss: 0.2902
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9625 - loss: 0.1196 - val_accuracy: 0.8761 - val_loss: 0.3229
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9644 - loss: 0.1061 - val_accuracy: 0.8841 - val_loss: 0.3269
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9771 - loss: 0.0791 - val_accuracy: 0.8737 - val_loss: 0.3815
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9785 - loss: 0.0714 - val_accuracy: 0.8811 - val_loss: 0.3720
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9868 - loss: 0.0525 - val_accuracy: 0.8778 - val_loss: 0.4006
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9902 - loss: 0.0441 - val_accuracy: 0.8699 - val_loss: 0.4425
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9895 - loss: 0.0424 - val_accuracy: 0.8749 - val_loss: 0.4538
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.9940 - loss: 0.0279 - val_accuracy: 0.8733 - val_loss: 0.4794
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9948 - loss: 0.0252 - val_accuracy: 0.8719 - val_loss: 0.5108
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9980 - loss: 0.0163 - val_accuracy: 0.8708 - val_loss: 0.5428
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9937 - loss: 0.0243 - val_accuracy: 0.8703 - val_loss: 0.5723
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9978 - loss: 0.0125 - val_accuracy: 0.8710 - val_loss: 0.6050
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9996 - loss: 0.0066 - val_accuracy: 0.8705 - val_loss: 0.6264
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9957 - loss: 0.0174 - val_accuracy: 0.8703 - val_loss: 0.6471
```

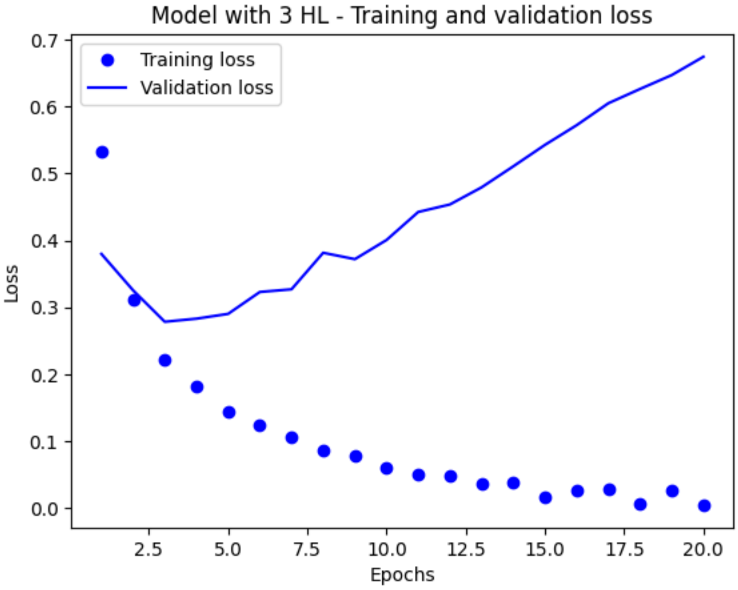
```
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9998 - loss: 0.0043 - val_accuracy: 0.8689 - val_loss: 0.6745
```

```
Model_3_Hidden_Layer_dict = Model_3_Hidden_Layer.history
Model_3_Hidden_Layer_dict.keys()
```

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

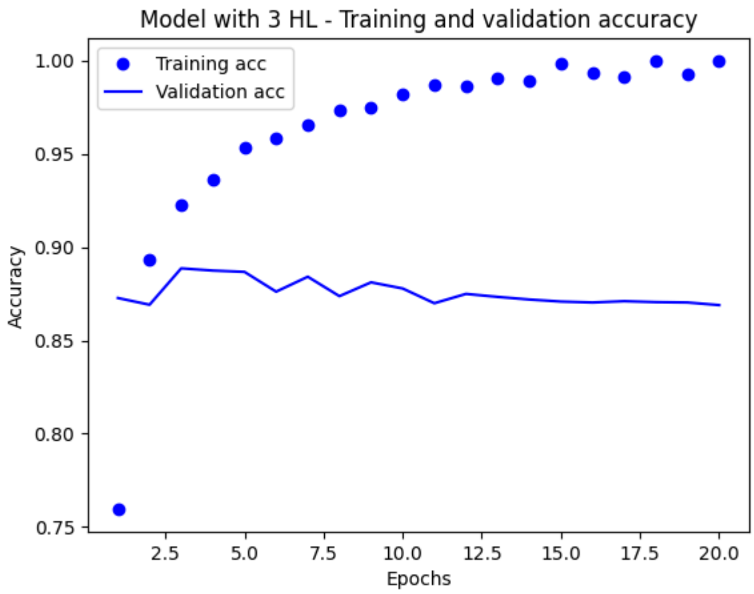
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_3_Hidden_Layer_dict = Model_3_Hidden_Layer.history
loss_values_3 = Model_3_Hidden_Layer_dict["loss"]
val_loss_values_3 = Model_3_Hidden_Layer_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_Hidden_Layer_dict["accuracy"]
val_acc_3 = Model_3_Hidden_Layer_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

```
model_3_HL = 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")
])
model_3_HL.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model_3_HL.fit(x_train, y_train, epochs=6, batch_size=512) # Epochs selected 6 because it starts to dip from 7
Model_3_Hidden_Layer_Results = model_3_HL.evaluate(x_test, y_test)
```

```
Epoch 1/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.6931 - loss: 0.5873
Epoch 2/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9028 - loss: 0.2814
Epoch 3/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9274 - loss: 0.2015
Epoch 4/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9420 - loss: 0.1661
Epoch 5/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9539 - loss: 0.1394
Epoch 6/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9576 - loss: 0.1277
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8747 - loss: 0.3424
```

```
Model_3_Hidden_Layer_Results
```

```
[0.3408409357070923, 0.8765199780464172]
```

Using Trained data to predict

```
model_3_HL.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.09414761],
       [0.99957156],
       [0.8042901 ],
       ...,
       [0.08851308],
       [0.02438793],
       [0.7443855 ]], dtype=float32)
```

3. Model With 32 Hidden Units

```
Model_32_Hidden_Units = model_32_HU.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

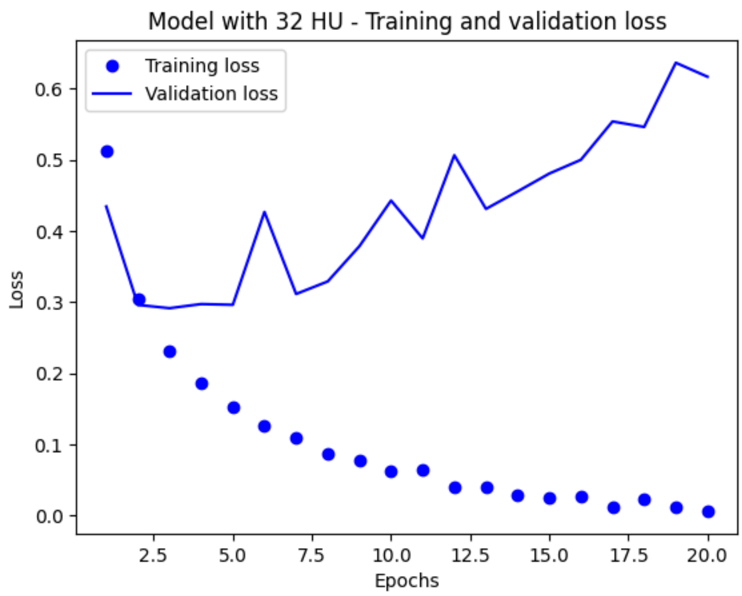
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 67ms/step - accuracy: 0.6825 - loss: 0.5921 - val_accuracy: 0.8046 - val_loss: 0.4342
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.8851 - loss: 0.3239 - val_accuracy: 0.8877 - val_loss: 0.2958
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 57ms/step - accuracy: 0.9200 - loss: 0.2308 - val_accuracy: 0.8834 - val_loss: 0.2913
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.9372 - loss: 0.1846 - val_accuracy: 0.8794 - val_loss: 0.2971
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9524 - loss: 0.1470 - val_accuracy: 0.8810 - val_loss: 0.2960
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.9634 - loss: 0.1185 - val_accuracy: 0.8450 - val_loss: 0.4265
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9645 - loss: 0.1092 - val_accuracy: 0.8837 - val_loss: 0.3112
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.9773 - loss: 0.0828 - val_accuracy: 0.8823 - val_loss: 0.3289
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step - accuracy: 0.9763 - loss: 0.0801 - val_accuracy: 0.8712 - val_loss: 0.3783
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9828 - loss: 0.0607 - val_accuracy: 0.8607 - val_loss: 0.4425
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9856 - loss: 0.0522 - val_accuracy: 0.8770 - val_loss: 0.3893
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 0.9940 - loss: 0.0336 - val_accuracy: 0.8564 - val_loss: 0.5063
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.9935 - loss: 0.0327 - val_accuracy: 0.8750 - val_loss: 0.4308
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.9975 - loss: 0.0217 - val_accuracy: 0.8732 - val_loss: 0.4553
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 57ms/step - accuracy: 0.9970 - loss: 0.0210 - val_accuracy: 0.8732 - val_loss: 0.4805
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.9971 - loss: 0.0178 - val_accuracy: 0.8758 - val_loss: 0.4998
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 42ms/step - accuracy: 0.9999 - loss: 0.0098 - val_accuracy: 0.8660 - val_loss: 0.5537
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - accuracy: 0.9930 - loss: 0.0259 - val_accuracy: 0.8724 - val_loss: 0.5460
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.9996 - loss: 0.0074 - val_accuracy: 0.8658 - val_loss: 0.6362
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - accuracy: 0.9997 - loss: 0.0078 - val_accuracy: 0.8668 - val_loss: 0.6166
```

```
Model_32_Hidden_Units_dict = Model_32_Hidden_Units.history
Model_32_Hidden_Units_dict.keys()
```

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

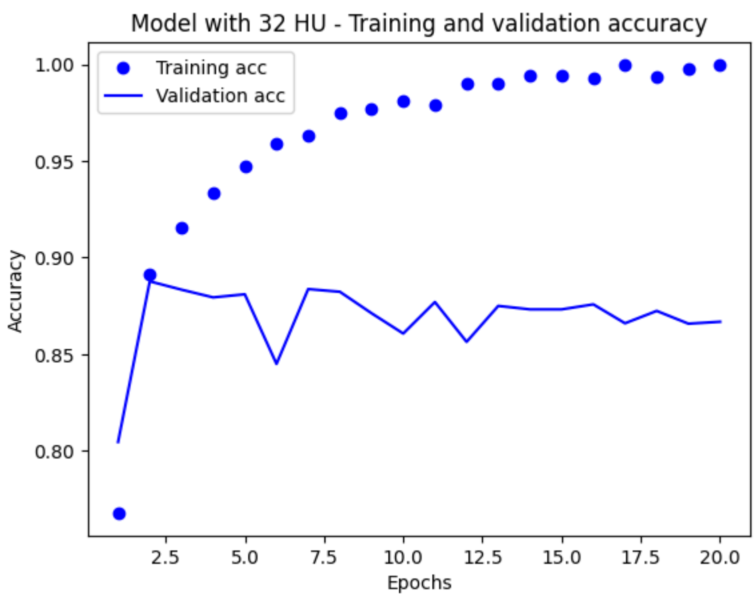
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_32_Hidden_Units_dict = Model_32_Hidden_Units.history
loss_values_32 = Model_32_Hidden_Units_dict["loss"]
val_loss_values_32 = Model_32_Hidden_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 = Model_32_Hidden_Units_dict["accuracy"]
val_acc_32 = Model_32_Hidden_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

```
model_32_HU = keras.Sequential([
    layers.Dense(32, activation="relu"), # 32 Hidden Units
    layers.Dense(32, activation="relu"), # 32 Hidden Units
    layers.Dense(1, activation="sigmoid")
])
model_32_HU.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model_32_HU.fit(x_train, y_train, epochs=3, batch_size=512) # Epochs selected 3 because it starts to dip from 3
Model_32_Hidden_Units_Results = model_32_HU.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 49ms/step - accuracy: 0.7290 - loss: 0.5515
Epoch 2/3
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 39ms/step - accuracy: 0.9003 - loss: 0.2751
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9229 - loss: 0.2076
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8874 - loss: 0.2792
```

```
Model_32_Hidden_Units_Results
```

```
[0.2781667709350586, 0.888759970664978]
```

Using Trained data to predict

```
model_32_HU.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.28948924],
       [0.99983054],
       [0.8771592 ],
       ...,
       [0.10240595],
       [0.06962123],
       [0.5774995 ]], dtype=float32)
```

4. Model With 64 Hidden Units

```
Model_64_Hidden_Units = model_64_HU.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

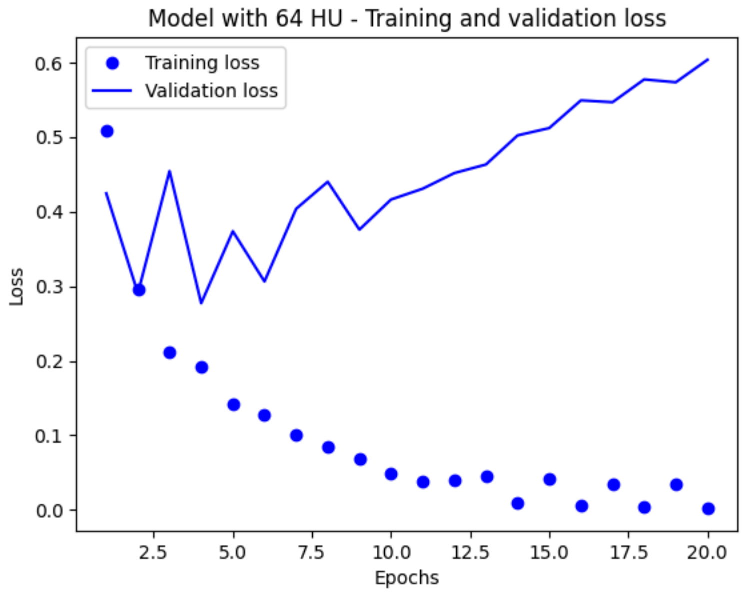
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 97ms/step - accuracy: 0.6944 - loss: 0.5893 - val_accuracy: 0.8124 - val_loss: 0.4243
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.8875 - loss: 0.3116 - val_accuracy: 0.8842 - val_loss: 0.2905
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - accuracy: 0.9265 - loss: 0.2129 - val_accuracy: 0.8179 - val_loss: 0.4542
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - accuracy: 0.9199 - loss: 0.2034 - val_accuracy: 0.8882 - val_loss: 0.2771
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 114ms/step - accuracy: 0.9542 - loss: 0.1389 - val_accuracy: 0.8633 - val_loss: 0.3735
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 61ms/step - accuracy: 0.9411 - loss: 0.1504 - val_accuracy: 0.8852 - val_loss: 0.3062
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - accuracy: 0.9645 - loss: 0.1061 - val_accuracy: 0.8684 - val_loss: 0.4036
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - accuracy: 0.9693 - loss: 0.0903 - val_accuracy: 0.8641 - val_loss: 0.4399
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 77ms/step - accuracy: 0.9780 - loss: 0.0654 - val_accuracy: 0.8811 - val_loss: 0.3757
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 93ms/step - accuracy: 0.9884 - loss: 0.0438 - val_accuracy: 0.8797 - val_loss: 0.4161
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 60ms/step - accuracy: 0.9888 - loss: 0.0398 - val_accuracy: 0.8760 - val_loss: 0.4306
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - accuracy: 0.9898 - loss: 0.0362 - val_accuracy: 0.8792 - val_loss: 0.4515
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - accuracy: 0.9932 - loss: 0.0272 - val_accuracy: 0.8775 - val_loss: 0.4630
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 122ms/step - accuracy: 0.9995 - loss: 0.0094 - val_accuracy: 0.8784 - val_loss: 0.5020
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.9946 - loss: 0.0218 - val_accuracy: 0.8773 - val_loss: 0.5118
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - accuracy: 0.9998 - loss: 0.0056 - val_accuracy: 0.8784 - val_loss: 0.5490
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9961 - loss: 0.0151 - val_accuracy: 0.8764 - val_loss: 0.5465
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 64ms/step - accuracy: 0.9999 - loss: 0.0038 - val_accuracy: 0.8777 - val_loss: 0.5770
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 116ms/step - accuracy: 0.9965 - loss: 0.0131 - val_accuracy: 0.8774 - val_loss: 0.5732
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 1.0000 - loss: 0.0027 - val_accuracy: 0.8767 - val_loss: 0.6034
```

```
Model_64_Hidden_Units_dict = Model_64_Hidden_Units.history
Model_64_Hidden_Units_dict.keys()
```

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

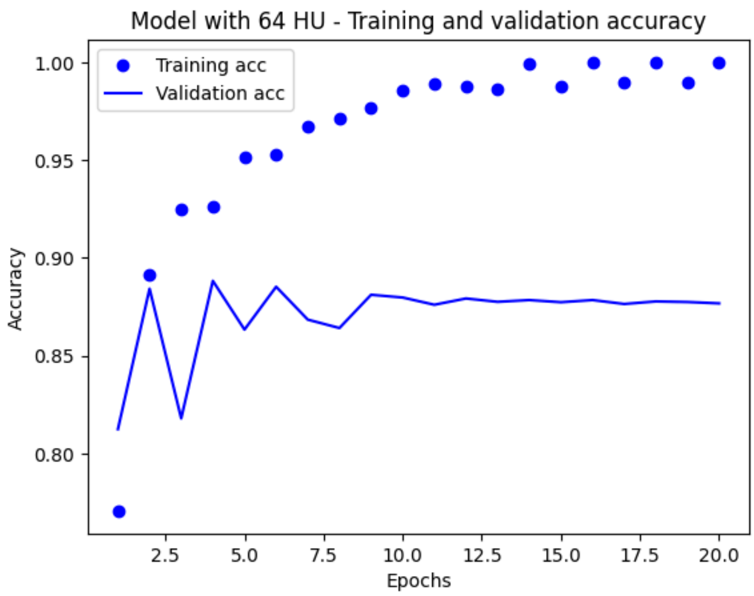
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_64_Hidden_Units_dict = Model_64_Hidden_Units.history
loss_values_64 = Model_64_Hidden_Units_dict["loss"]
val_loss_values_64 = Model_64_Hidden_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 = Model_64_Hidden_Units_dict["accuracy"]
val_acc_64 = Model_64_Hidden_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

```
model_64_HU = keras.Sequential([
    layers.Dense(64, activation="relu"), # 64 Hidden Units
    layers.Dense(64, activation="relu"), # 64 Hidden Units
    layers.Dense(1, activation="sigmoid")
])
model_64_HU.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model_64_HU.fit(x_train, y_train, epochs=2, batch_size=512) # Epochs selected 2 because it starts to dip from 2
Model_64_Hidden_Units_Results = model_64_HU.evaluate(x_test, y_test)
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 45ms/step - accuracy: 0.7241 - loss: 0.5456
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 56ms/step - accuracy: 0.9010 - loss: 0.2670
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8571 - loss: 0.3473
```

```
Model_64_Hidden_Units_Results
```

```
[0.3409341275691986, 0.8587599992752075]
```

Using Trained data to predict

```
model_64_HU.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
array([[0.39997244],
       [0.99975413],
       [0.9876823 ],
       ...,
       [0.18779352],
       [0.24800734],
       [0.8131342 ]], dtype=float32)
```

5. Model With MSE Loss

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

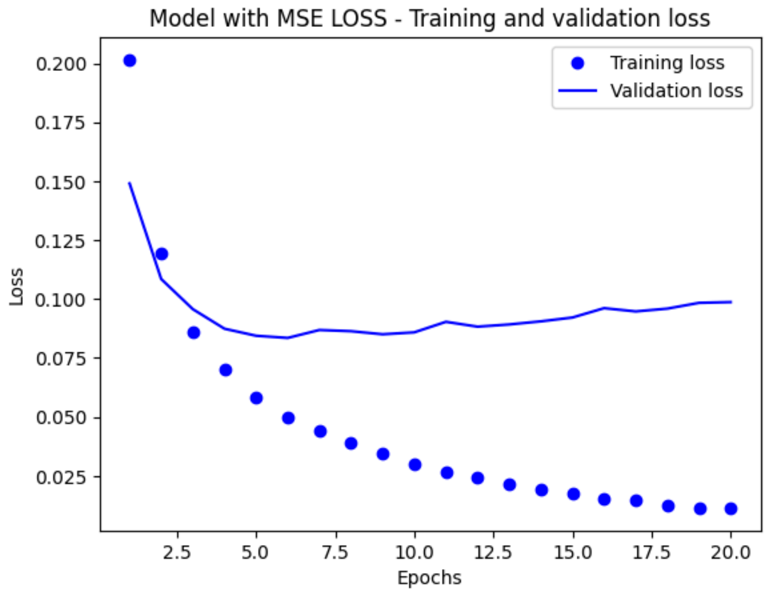
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 76ms/step - accuracy: 0.6206 - loss: 0.2267 - val_accuracy: 0.8428 - val_loss: 0.1490
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.8766 - loss: 0.1282 - val_accuracy: 0.8749 - val_loss: 0.1085
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9127 - loss: 0.0867 - val_accuracy: 0.8792 - val_loss: 0.0957
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9249 - loss: 0.0708 - val_accuracy: 0.8870 - val_loss: 0.0874
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9374 - loss: 0.0577 - val_accuracy: 0.8865 - val_loss: 0.0844
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9506 - loss: 0.0498 - val_accuracy: 0.8867 - val_loss: 0.0835
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9545 - loss: 0.0433 - val_accuracy: 0.8817 - val_loss: 0.0869
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9632 - loss: 0.0365 - val_accuracy: 0.8805 - val_loss: 0.0864
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9670 - loss: 0.0348 - val_accuracy: 0.8828 - val_loss: 0.0851
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.9733 - loss: 0.0292 - val_accuracy: 0.8828 - val_loss: 0.0859
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.9790 - loss: 0.0249 - val_accuracy: 0.8784 - val_loss: 0.0904
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9800 - loss: 0.0237 - val_accuracy: 0.8817 - val_loss: 0.0883
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9845 - loss: 0.0198 - val_accuracy: 0.8808 - val_loss: 0.0892
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9857 - loss: 0.0181 - val_accuracy: 0.8794 - val_loss: 0.0905
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9870 - loss: 0.0165 - val_accuracy: 0.8780 - val_loss: 0.0922
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9894 - loss: 0.0146 - val_accuracy: 0.8745 - val_loss: 0.0961
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9894 - loss: 0.0132 - val_accuracy: 0.8763 - val_loss: 0.0948
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9915 - loss: 0.0116 - val_accuracy: 0.8748 - val_loss: 0.0959
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9915 - loss: 0.0113 - val_accuracy: 0.8735 - val_loss: 0.0984
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9910 - loss: 0.0111 - val_accuracy: 0.8729 - val_loss: 0.0987
```

```
Model_MSE_LOSS_dict = Model_MSE_LOSS.history
Model_MSE_LOSS_dict.keys()
```

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

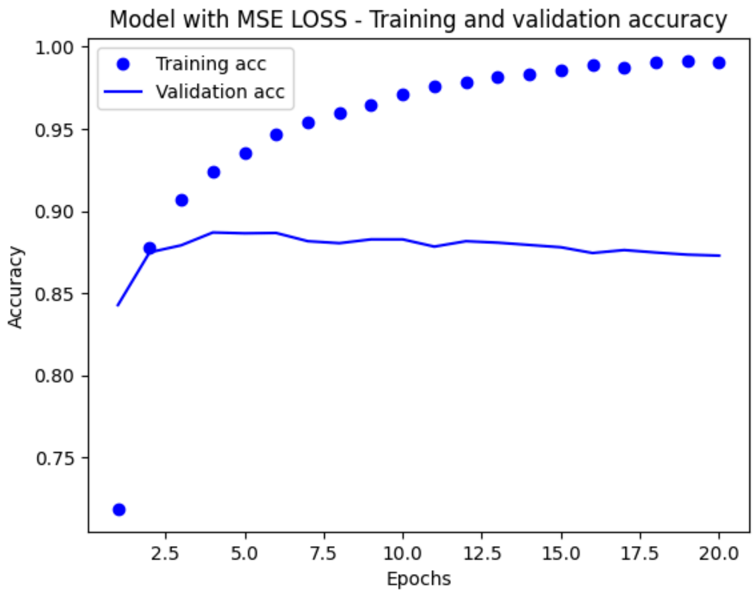
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_MSE_LOSS_dict = Model_MSE_LOSS.history
loss_values_MSE = Model_MSE_LOSS_dict["loss"]
val_loss_values_MSE = Model_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()
```



Plotting Accuracy

```
plt.clf()
acc_MSE = Model_MSE_LOSS_dict["accuracy"]
val_acc_MSE = Model_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

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

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 25ms/step - accuracy: 0.7240 - loss: 0.2045
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.8905 - loss: 0.0997
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 37ms/step - accuracy: 0.9142 - loss: 0.0727
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.9312 - loss: 0.0597
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8823 - loss: 0.0868
```

```
Model_MSE_LOSS_Results
```

```
[0.086273193359375, 0.8833600282669067]
```

Using Trained data to predict

```
model_mse.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.17323162],
       [0.9993285 ],
       [0.8196411 ],
       ...,
       [0.12254603],
       [0.11084169],
       [0.40304676]], dtype=float32)
```

6. Model With tanh activation

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

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 84ms/step - accuracy: 0.7070 - loss: 0.5672 - val_accuracy: 0.8417 - val_loss: 0.3885
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8914 - loss: 0.3095 - val_accuracy: 0.8772 - val_loss: 0.3006
Epoch 3/20
```

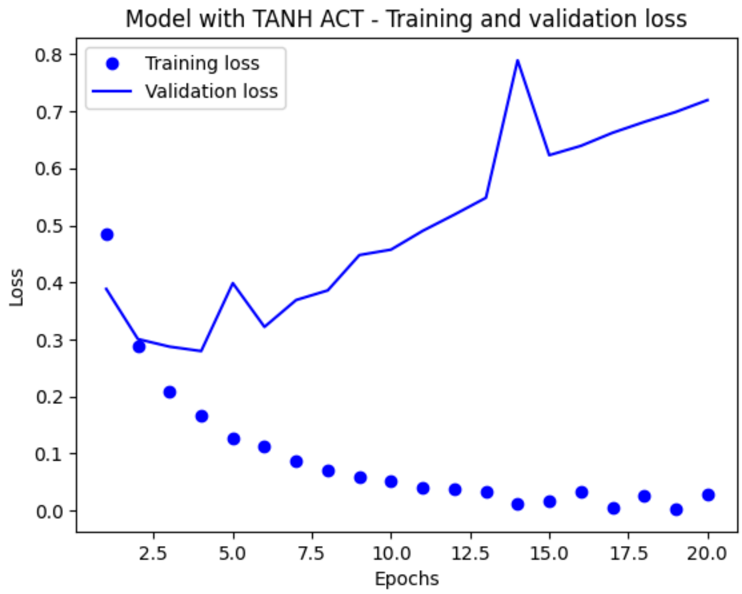
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9249 - loss: 0.2159 - val_accuracy: 0.8820 - val_loss: 0.2873
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9433 - loss: 0.1705 - val_accuracy: 0.8857 - val_loss: 0.2795
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9590 - loss: 0.1247 - val_accuracy: 0.8569 - val_loss: 0.3986
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9620 - loss: 0.1117 - val_accuracy: 0.8818 - val_loss: 0.3219
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9778 - loss: 0.0765 - val_accuracy: 0.8720 - val_loss: 0.3690
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9801 - loss: 0.0658 - val_accuracy: 0.8766 - val_loss: 0.3859
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9855 - loss: 0.0493 - val_accuracy: 0.8707 - val_loss: 0.4479
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9858 - loss: 0.0463 - val_accuracy: 0.8724 - val_loss: 0.4574
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9898 - loss: 0.0342 - val_accuracy: 0.8708 - val_loss: 0.4903
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9953 - loss: 0.0239 - val_accuracy: 0.8703 - val_loss: 0.5187
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9968 - loss: 0.0167 - val_accuracy: 0.8672 - val_loss: 0.5483
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9992 - loss: 0.0099 - val_accuracy: 0.8339 - val_loss: 0.7892
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9884 - loss: 0.0334 - val_accuracy: 0.8611 - val_loss: 0.6230
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9959 - loss: 0.0182 - val_accuracy: 0.8639 - val_loss: 0.6393
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9996 - loss: 0.0044 - val_accuracy: 0.8630 - val_loss: 0.6622
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9974 - loss: 0.0114 - val_accuracy: 0.8636 - val_loss: 0.6811
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9999 - loss: 0.0027 - val_accuracy: 0.8641 - val_loss: 0.6987
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - accuracy: 0.9982 - loss: 0.0081 - val_accuracy: 0.8644 - val_loss: 0.7195
```

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

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

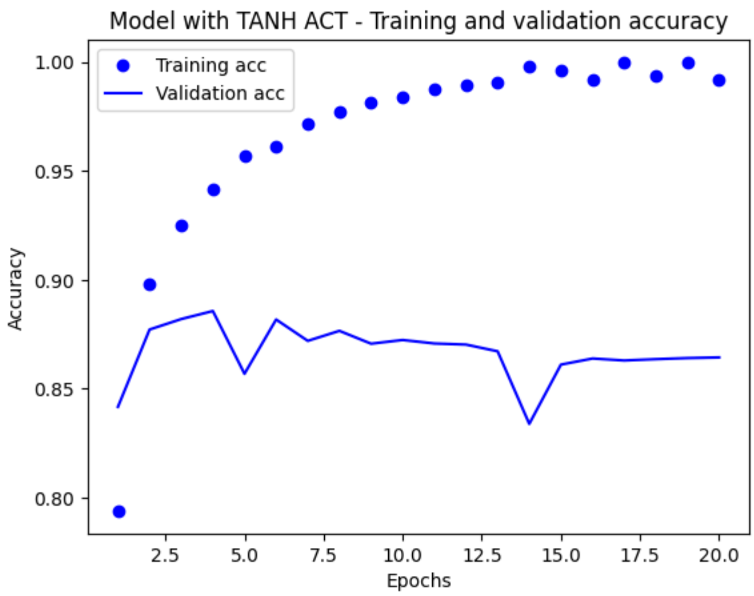
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_TANH_ACT_dict = Model_TANH_ACT.history
loss_values_TANH = Model_TANH_ACT_dict["loss"]
val_loss_values_TANH = Model_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 = Model_TANH_ACT_dict["accuracy"]
val_acc_TANH = Model_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 = keras.Sequential([
    layers.Dense(16, activation="tanh"), # tanh activation
    layers.Dense(16, activation="tanh"), # tanh activation
    layers.Dense(1, activation="sigmoid")
])
model_tanh.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model_tanh.fit(x_train, y_train, epochs=3, batch_size=512) # Epochs selected 3 because it starts to dip from 3
Model_TANH_ACT_Results = model_tanh.evaluate(x_test, y_test)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.7369 - loss: 0.5302
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9085 - loss: 0.2583
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9310 - loss: 0.1901
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8728 - loss: 0.3160
```

```
Model_TANH_ACT_Results
```

```
[0.31786277890205383, 0.872160017490387]
```

Using Trained data to predict

```
model_tanh.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.09332374],
       [0.9956134 ],
       [0.24930619],
       ...,
       [0.04044585],
       [0.03484334],
       [0.5587447 ]], dtype=float32)
```

7. Model With L2 Regularization

```
Model_Reg_Tech = model_reg.fit(partial_x_train,
```

```
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

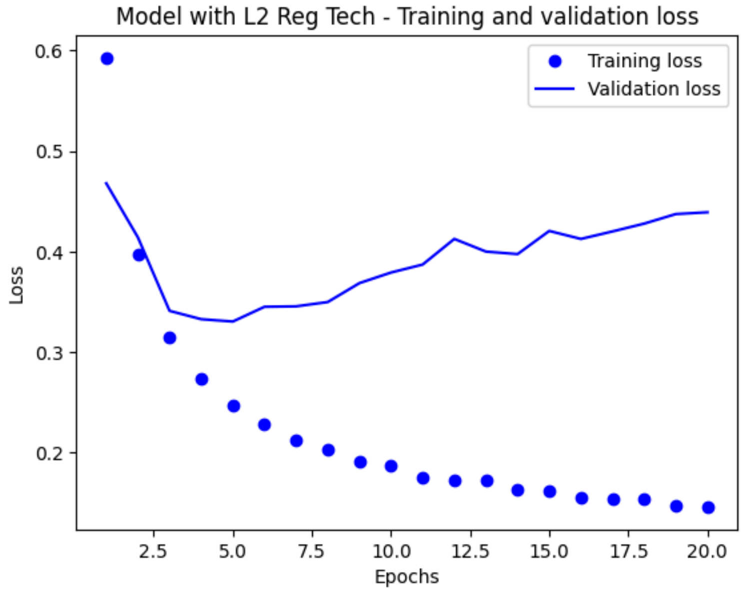
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 62ms/step - accuracy: 0.6884 - loss: 0.6590 - val_accuracy: 0.8615 - val_loss: 0.4679
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8834 - loss: 0.4197 - val_accuracy: 0.8478 - val_loss: 0.4141
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 53ms/step - accuracy: 0.9079 - loss: 0.3238 - val_accuracy: 0.8866 - val_loss: 0.3410
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9250 - loss: 0.2757 - val_accuracy: 0.8860 - val_loss: 0.3327
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9358 - loss: 0.2456 - val_accuracy: 0.8865 - val_loss: 0.3303
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9493 - loss: 0.2192 - val_accuracy: 0.8804 - val_loss: 0.3450
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9541 - loss: 0.2055 - val_accuracy: 0.8816 - val_loss: 0.3455
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9580 - loss: 0.1950 - val_accuracy: 0.8828 - val_loss: 0.3497
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9625 - loss: 0.1851 - val_accuracy: 0.8773 - val_loss: 0.3685
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9629 - loss: 0.1818 - val_accuracy: 0.8796 - val_loss: 0.3791
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9712 - loss: 0.1681 - val_accuracy: 0.8744 - val_loss: 0.3871
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9669 - loss: 0.1712 - val_accuracy: 0.8724 - val_loss: 0.4126
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step - accuracy: 0.9651 - loss: 0.1702 - val_accuracy: 0.8726 - val_loss: 0.3999
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9736 - loss: 0.1546 - val_accuracy: 0.8762 - val_loss: 0.3975
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9744 - loss: 0.1566 - val_accuracy: 0.8718 - val_loss: 0.4205
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9751 - loss: 0.1510 - val_accuracy: 0.8748 - val_loss: 0.4126
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9765 - loss: 0.1480 - val_accuracy: 0.8742 - val_loss: 0.4200
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9775 - loss: 0.1468 - val_accuracy: 0.8742 - val_loss: 0.4278
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9821 - loss: 0.1366 - val_accuracy: 0.8729 - val_loss: 0.4373
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9819 - loss: 0.1364 - val_accuracy: 0.8742 - val_loss: 0.4391
```

```
Model_Reg_Tech_dict = Model_Reg_Tech.history
Model_Reg_Tech_dict.keys()
```

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

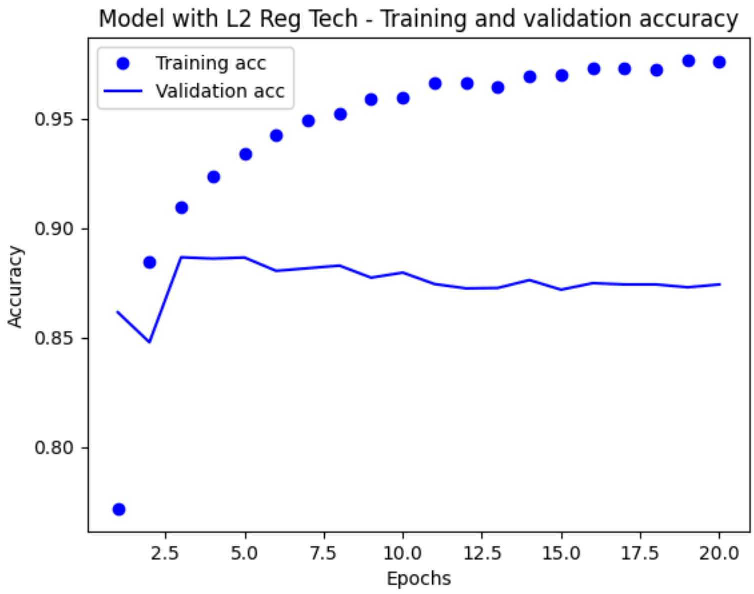
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_Reg_Tech_dict = Model_Reg_Tech.history
loss_values_Reg = Model_Reg_Tech_dict["loss"]
val_loss_values_Reg = Model_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 = Model_Reg_Tech_dict["accuracy"]
val_acc_Reg = Model_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

```
model_reg = keras.Sequential([
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)), # Applied L2 regularization (0.001 - common acceptec
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)), # Applied L2 regularization (0.001 - common acceptec
    layers.Dense(1, activation="sigmoid")
])
model_reg.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model_reg.fit(x_train, y_train, epochs=2, batch_size=512) # Epochs selected 2 because it starts to dip from 3
Model_Reg_Tech_Results = model_reg.evaluate(x_test, y_test)
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 28ms/step - accuracy: 0.7198 - loss: 0.6214
Epoch 2/2
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 28ms/step - accuracy: 0.9010 - loss: 0.3510
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8859 - loss: 0.3442
```

```
Model_Reg_Tech_Results
```

```
[0.34400734305381775, 0.8865600228309631]
```

Using Trained data to predict

```
model_reg.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
array([[0.32413855],
       [0.99838555],
       [0.75882363],
       ...,
       [0.1789023 ],
       [0.18363708],
       [0.5152261 ]], dtype=float32)
```

8. Model With Dropout Technique`

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

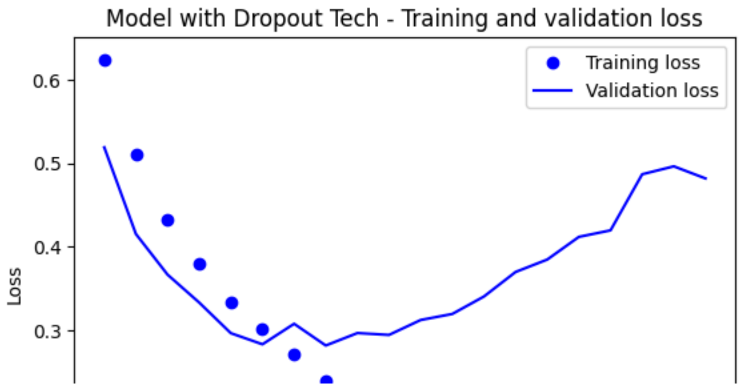
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 66ms/step - accuracy: 0.5922 - loss: 0.6562 - val_accuracy: 0.8539 - val_loss: 0.5190
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.7580 - loss: 0.5248 - val_accuracy: 0.8697 - val_loss: 0.4152
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.8315 - loss: 0.4410 - val_accuracy: 0.8717 - val_loss: 0.3666
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8637 - loss: 0.3843 - val_accuracy: 0.8823 - val_loss: 0.3331
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.8858 - loss: 0.3429 - val_accuracy: 0.8886 - val_loss: 0.2964
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.9071 - loss: 0.3043 - val_accuracy: 0.8883 - val_loss: 0.2831
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 63ms/step - accuracy: 0.9168 - loss: 0.2801 - val_accuracy: 0.8834 - val_loss: 0.3077
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9242 - loss: 0.2467 - val_accuracy: 0.8884 - val_loss: 0.2817
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9387 - loss: 0.2134 - val_accuracy: 0.8879 - val_loss: 0.2966
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9442 - loss: 0.1968 - val_accuracy: 0.8837 - val_loss: 0.2944
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9451 - loss: 0.1841 - val_accuracy: 0.8868 - val_loss: 0.3122
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9548 - loss: 0.1573 - val_accuracy: 0.8833 - val_loss: 0.3195
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9575 - loss: 0.1416 - val_accuracy: 0.8830 - val_loss: 0.3404
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.9601 - loss: 0.1320 - val_accuracy: 0.8844 - val_loss: 0.3698
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9645 - loss: 0.1209 - val_accuracy: 0.8832 - val_loss: 0.3846
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.9694 - loss: 0.1063 - val_accuracy: 0.8835 - val_loss: 0.4118
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9688 - loss: 0.1054 - val_accuracy: 0.8838 - val_loss: 0.4196
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9714 - loss: 0.0960 - val_accuracy: 0.8822 - val_loss: 0.4868
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9751 - loss: 0.0868 - val_accuracy: 0.8809 - val_loss: 0.4963
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9765 - loss: 0.0816 - val_accuracy: 0.8802 - val_loss: 0.4818
```

```
Model_Drp_Tech_dict = Model_Drp_Tech.history
Model_Drp_Tech_dict.keys()
```

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

Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_Drp_Tech_dict = Model_Drp_Tech.history
loss_values_Drp = Model_Drp_Tech_dict["loss"]
val_loss_values_Drp = Model_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()
```

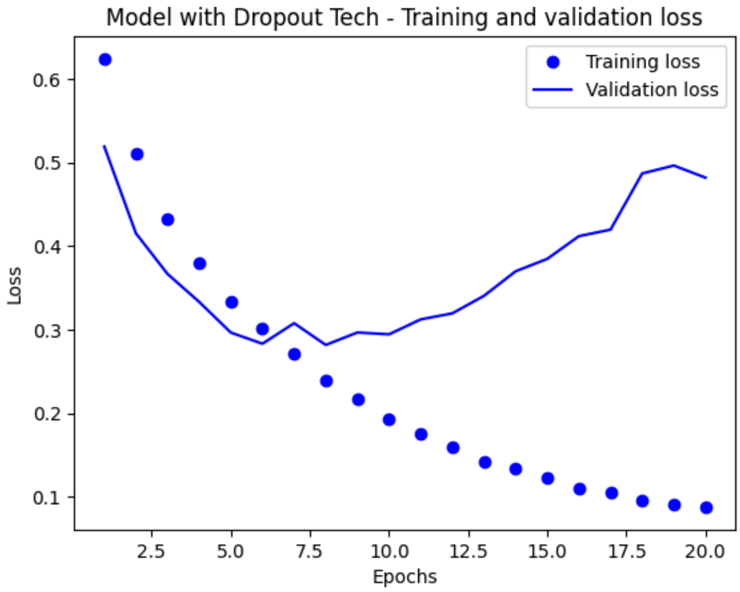


```
Model_Drp_Tech_dict = Model_Drp_Tech.history
Model_Drp_Tech_dict.keys()
```

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

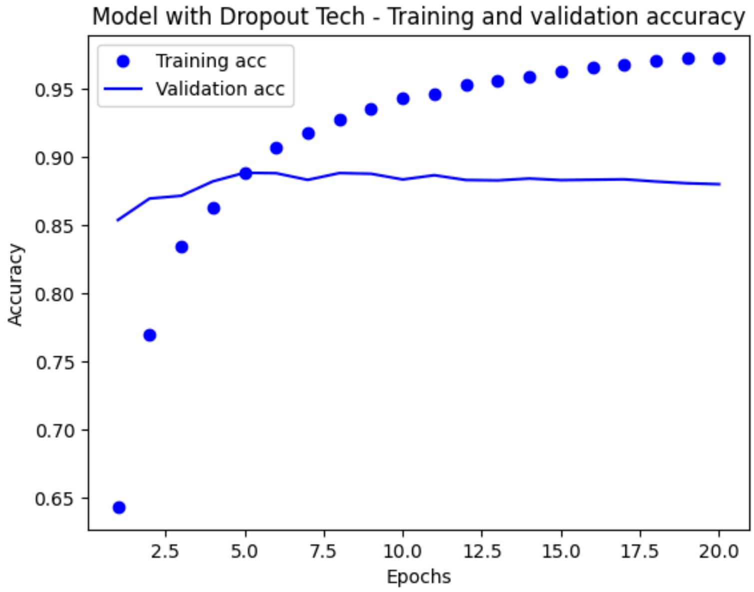
Plotting the graphshowing training and validation loss

```
import matplotlib.pyplot as plt
Model_Drp_Tech_dict = Model_Drp_Tech.history
loss_values_Drp = Model_Drp_Tech_dict["loss"]
val_loss_values_Drp = Model_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 = Model_Drp_Tech_dict["accuracy"]
val_acc_Drp = Model_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

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

```
Epoch 1/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.6007 - loss: 0.6538
Epoch 2/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 41ms/step - accuracy: 0.7926 - loss: 0.5020
Epoch 3/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.8597 - loss: 0.3998
Epoch 4/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - accuracy: 0.8935 - loss: 0.3324
Epoch 5/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9057 - loss: 0.2805
Epoch 6/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 26ms/step - accuracy: 0.9225 - loss: 0.2462
Epoch 7/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 34ms/step - accuracy: 0.9269 - loss: 0.2200
Epoch 8/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.9380 - loss: 0.1955
Epoch 9/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9398 - loss: 0.1834
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8779 - loss: 0.3546
```

```
Model_Drp_Tech_Results
```

```
[0.34334075450897217, 0.8809599876403809]
```

Using Trained data to predict

```
model_drp.predict(x_test)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.11506828],
       [1.        ],
       [0.99996316],
       ...,
       [0.10267788],
       [0.08504791],
       [0.81308633]], dtype=float32)
```

Comparison of the Models

Retrieveing the training history for all models (For Organising)

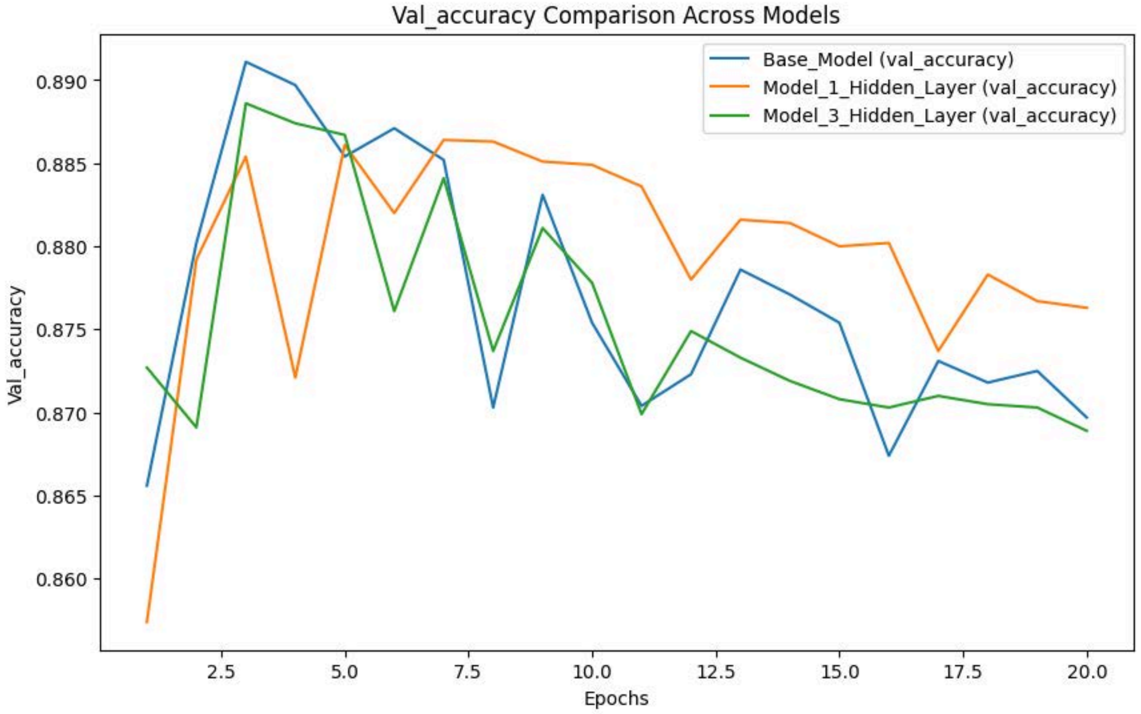
```
Base_model_dict = Base_model.history
Base_model_dict.keys()

Model_1_Hidden_Layer_dict = Model_1_Hidden_Layer.history
Model_1_Hidden_Layer_dict.keys()

Model_3_Hidden_Layer_dict = Model_3_Hidden_Layer.history
Model_3_Hidden_Layer_dict.keys()

Model_32_Hidden_Units_dict = Model_32_Hidden_Units.history
Model_32_Hidden_Units_dict.keys()

Model_64_Hidden_Units_dict = Model_64_Hidden_Units.history
Model_64_Hidden_Units_dict.keys()

Model_MSE_LOSS_dict = Model_MSE_LOSS.history
Model_MSE_LOSS_dict.keys()

Model_TANH_ACT_dict = Model_TANH_ACT.history
Model_TANH_ACT_dict.keys()

Model_Reg_Tech_dict = Model_Reg_Tech.history
Model_Reg_Tech_dict.keys()

Model_Drp_Tech_dict = Model_Drp_Tech.history
Model_Drp_Tech_dict.keys()
```

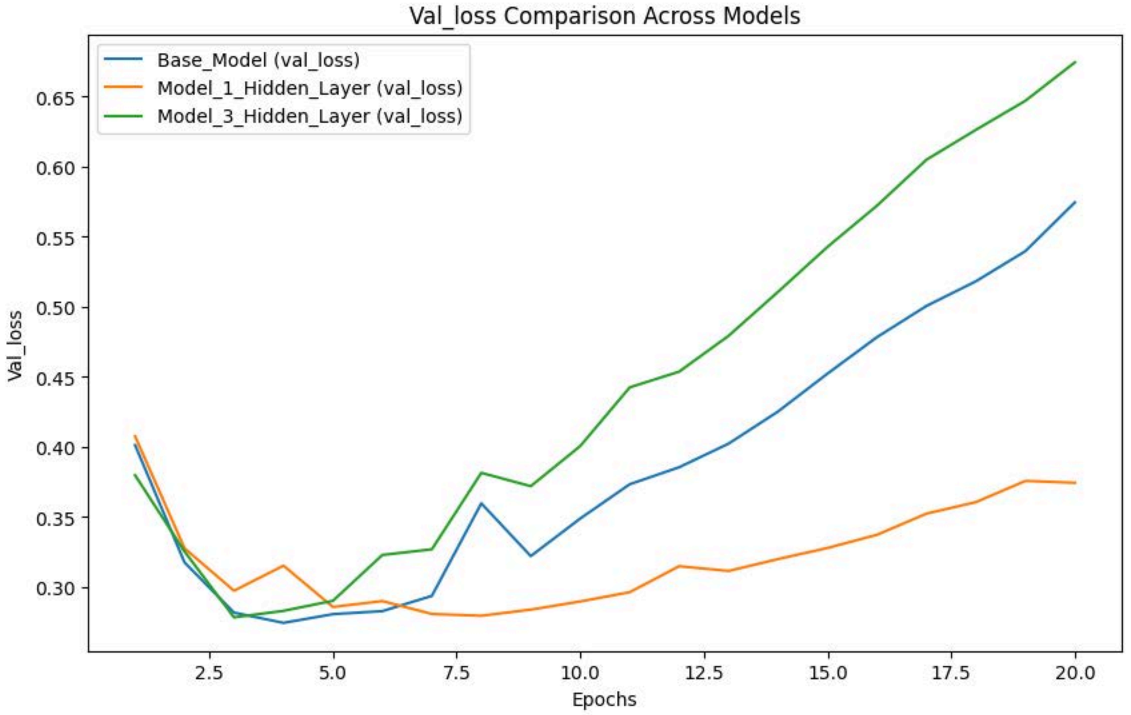
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

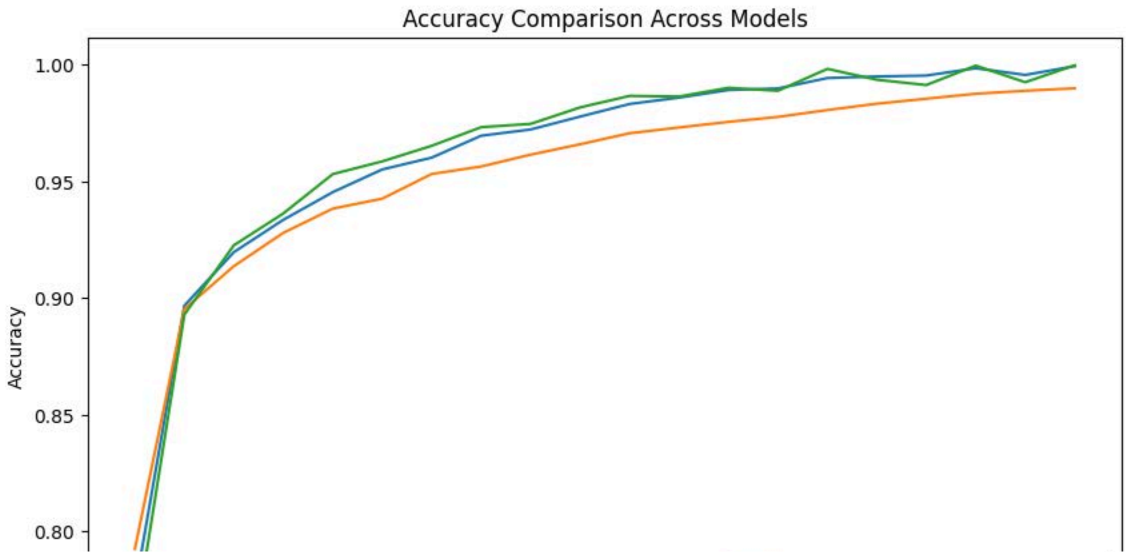
Question 1 - Comparing Hidden layers with Base Model

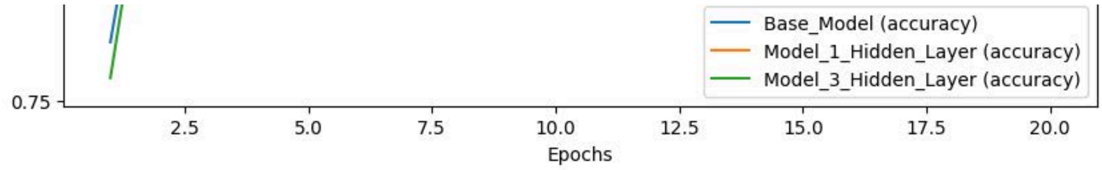
```
import matplotlib.pyplot as plt

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

# 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')
```

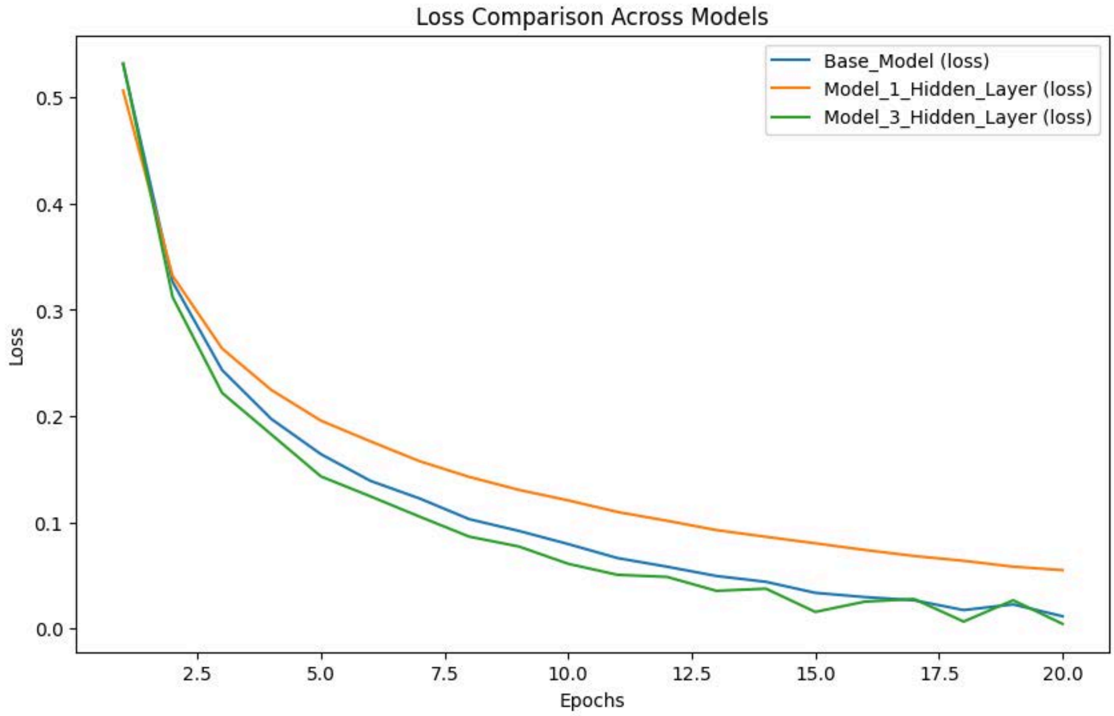
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_1_Hidden_Layer history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_3_Hidden_Layer history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







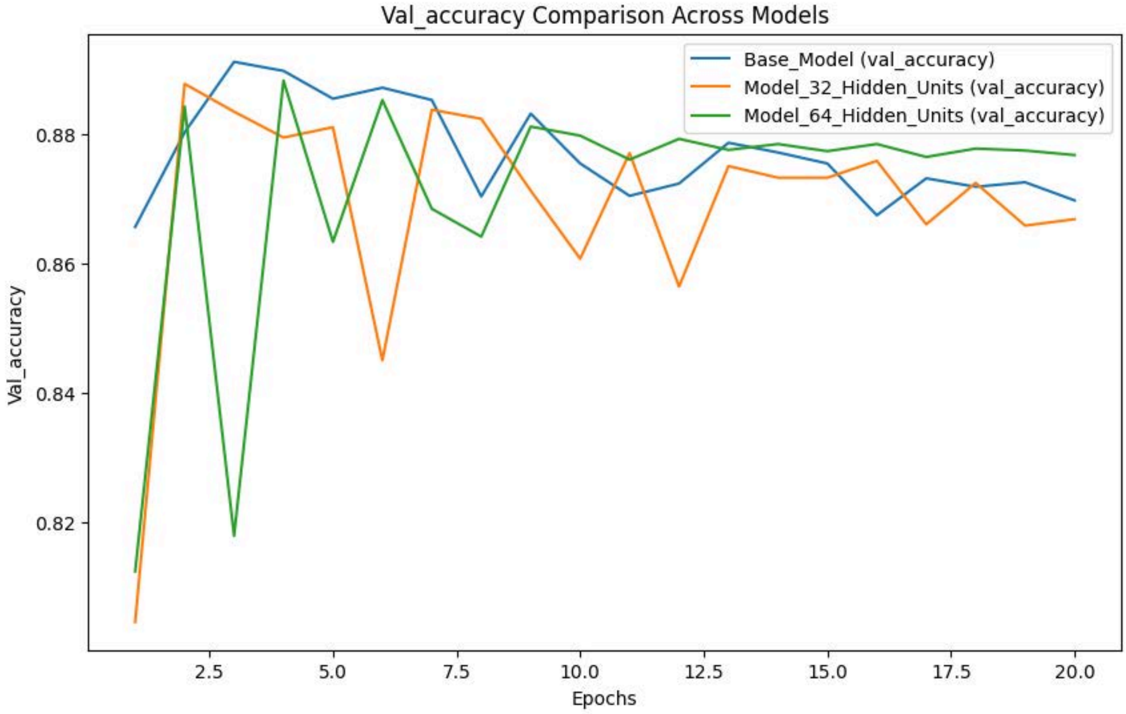


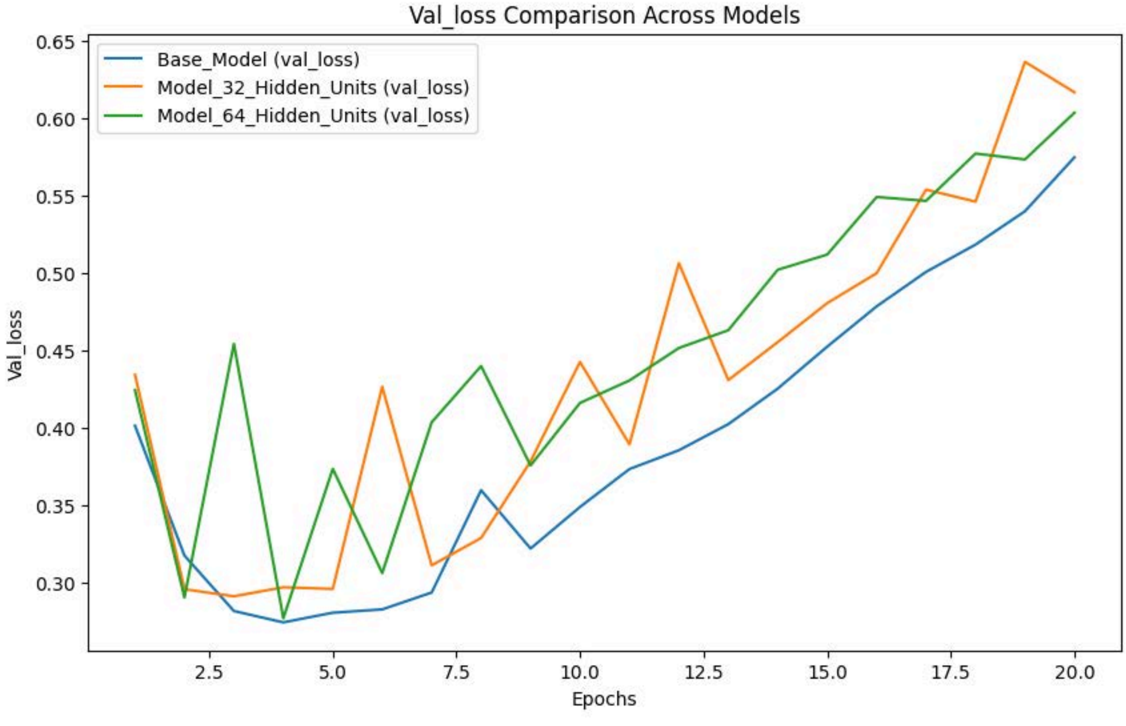


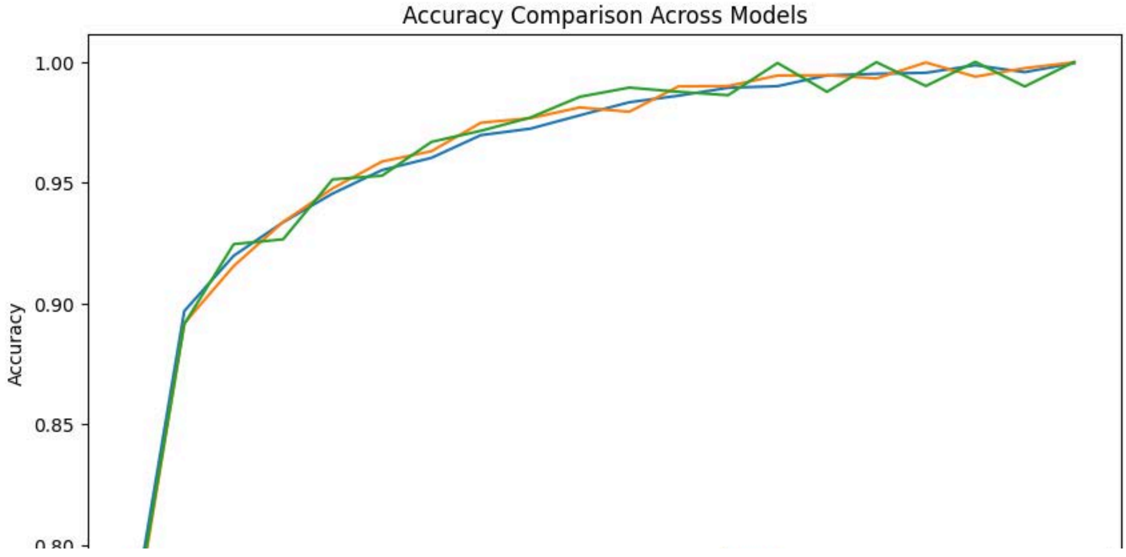
Question 2 - Comparing Base model with Hidden Units value of 16, 32 and 64

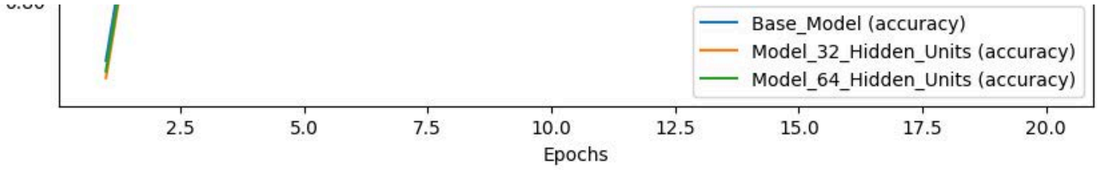
```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": Base_model,
    "Model_32_Hidden_Units": Model_32_Hidden_Units,
    "Model_64_Hidden_Units": Model_64_Hidden_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')
```

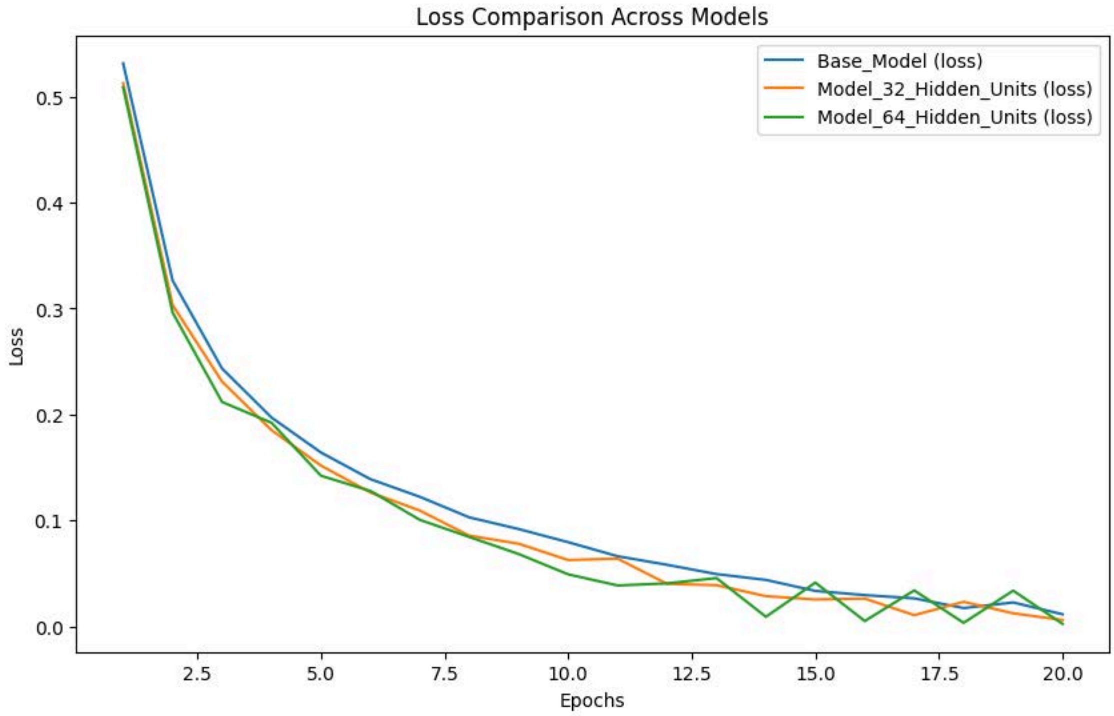
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_32_Hidden_Units history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_64_Hidden_Units history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







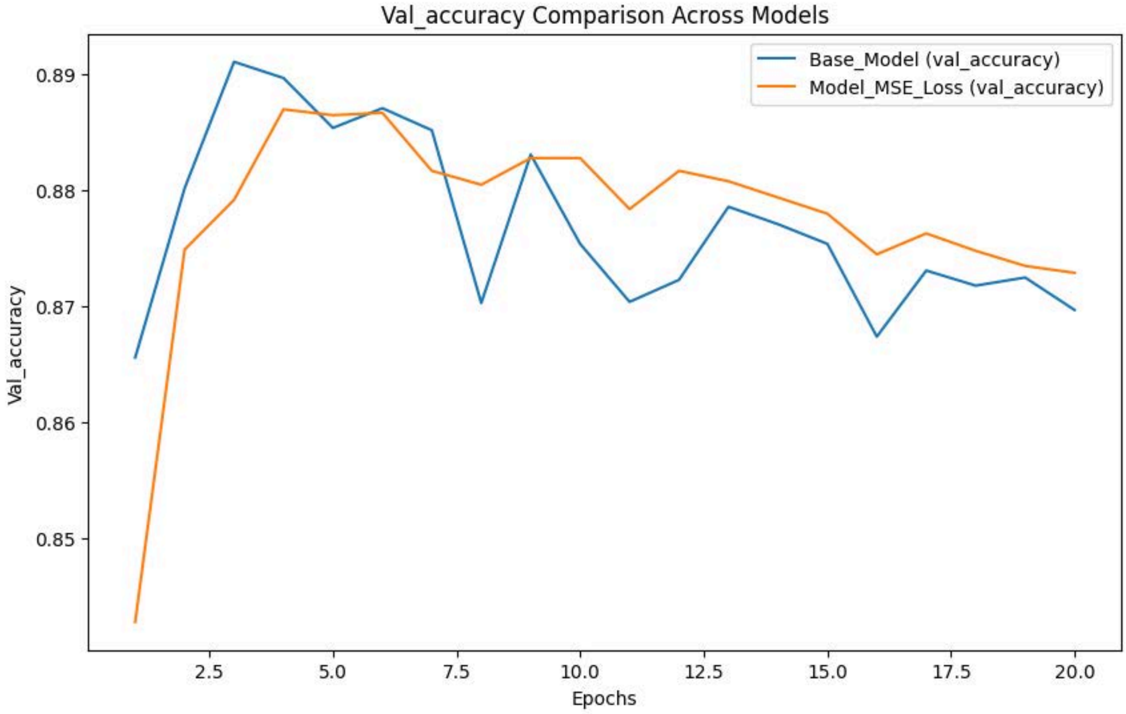


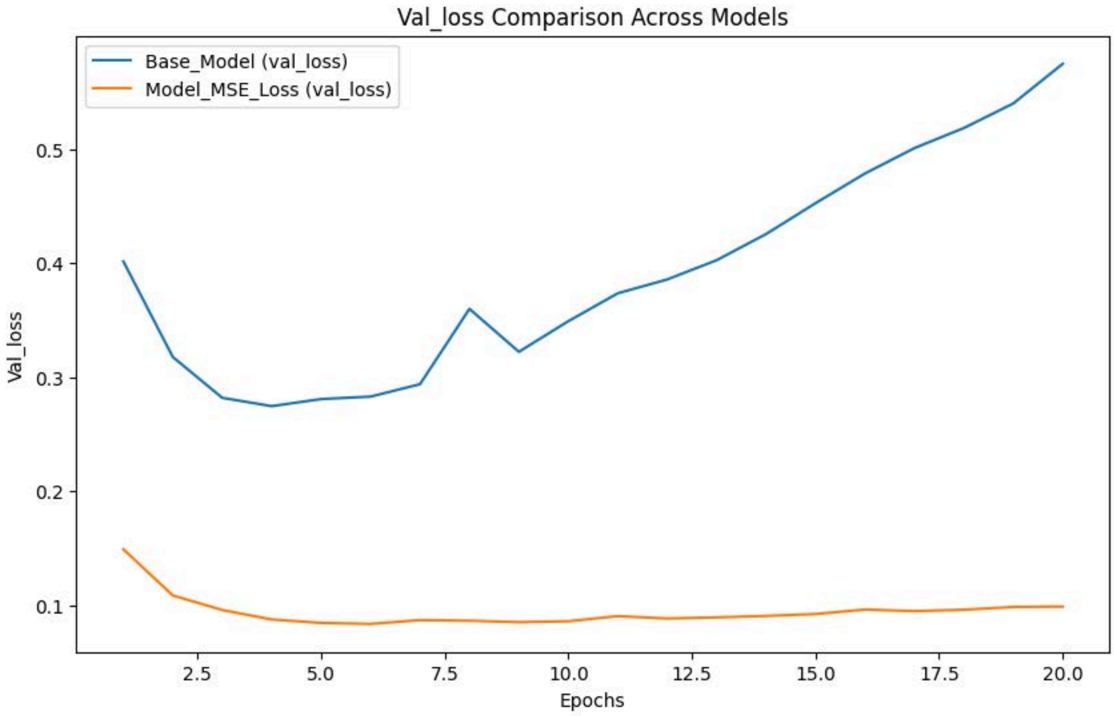


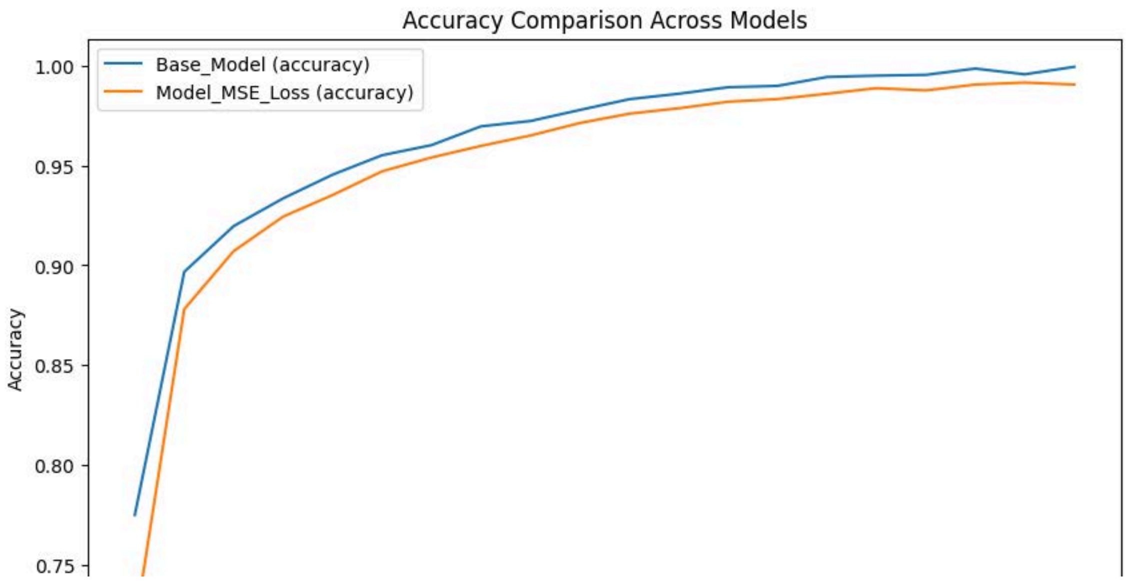
Question 3 - Comparing of MSE loss function

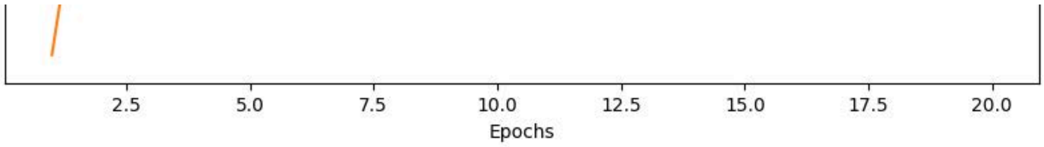
```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": Base_model,
    "Model_MSE_Loss": Model_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')
```

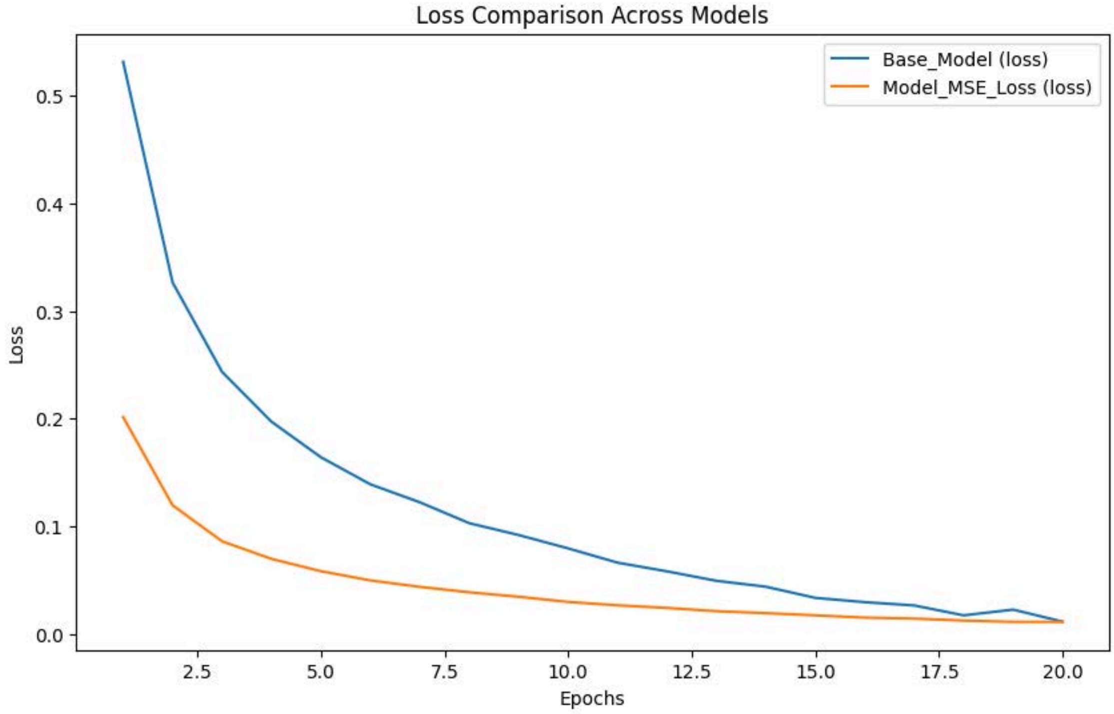
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_MSE_Loss history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







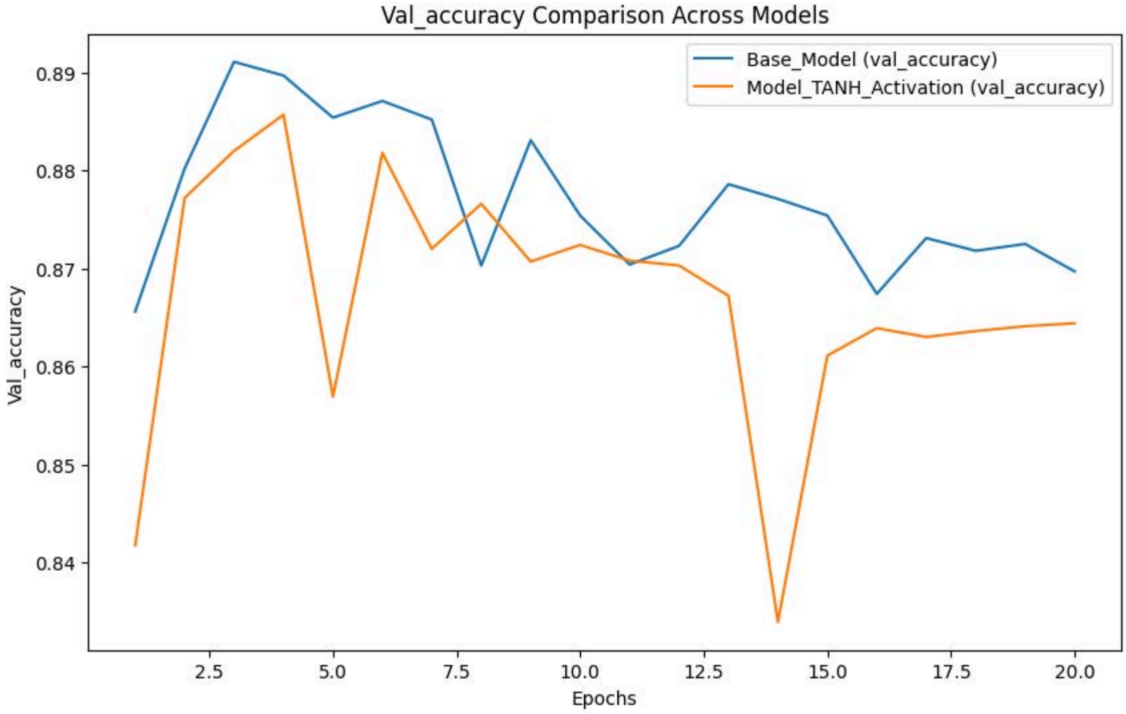


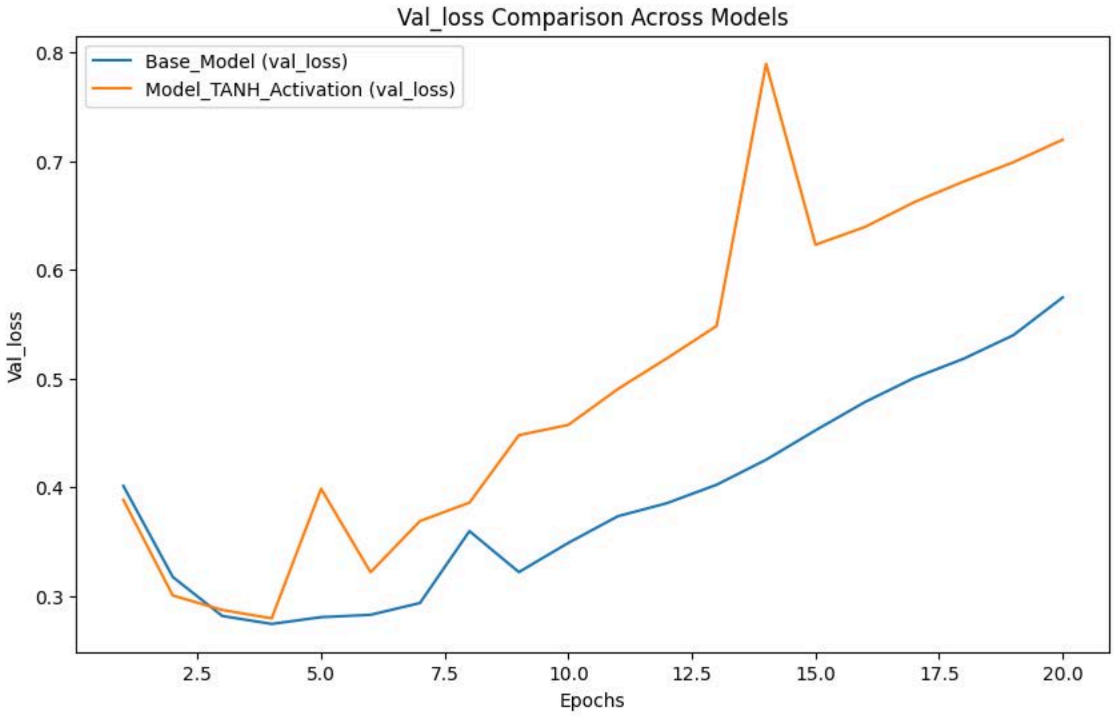


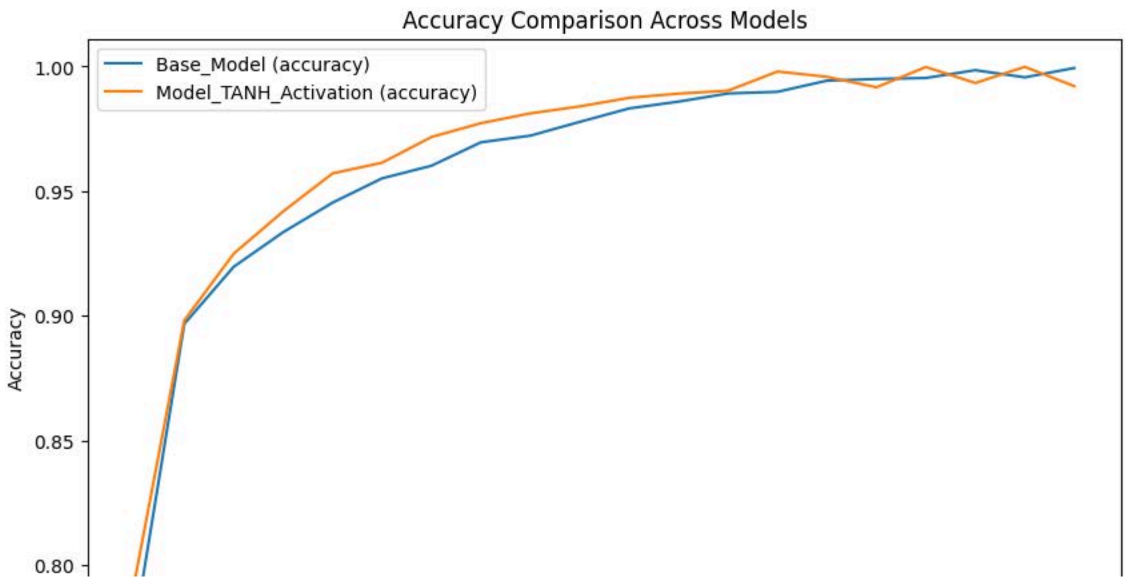
Question 4 - Comparing of Tanh activation with base model

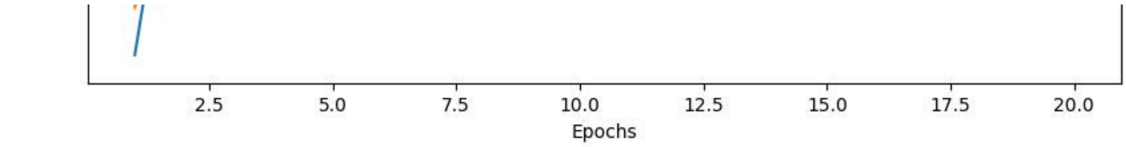
```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": Base_model,
    "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')
```

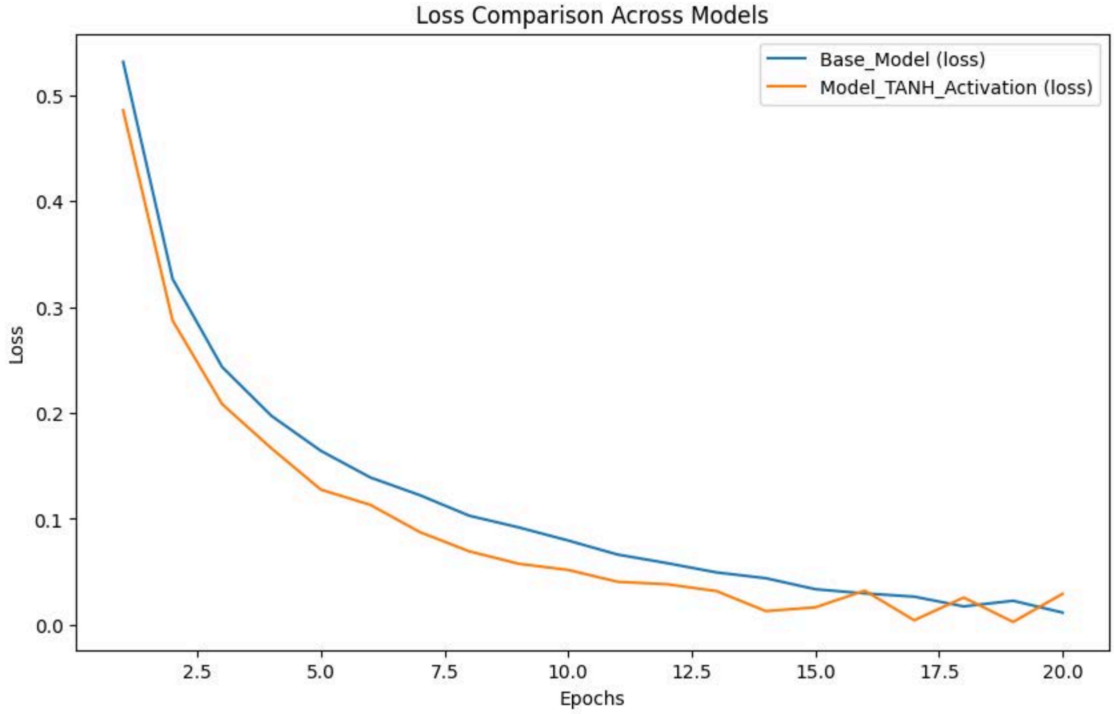
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_TANH_Activation history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







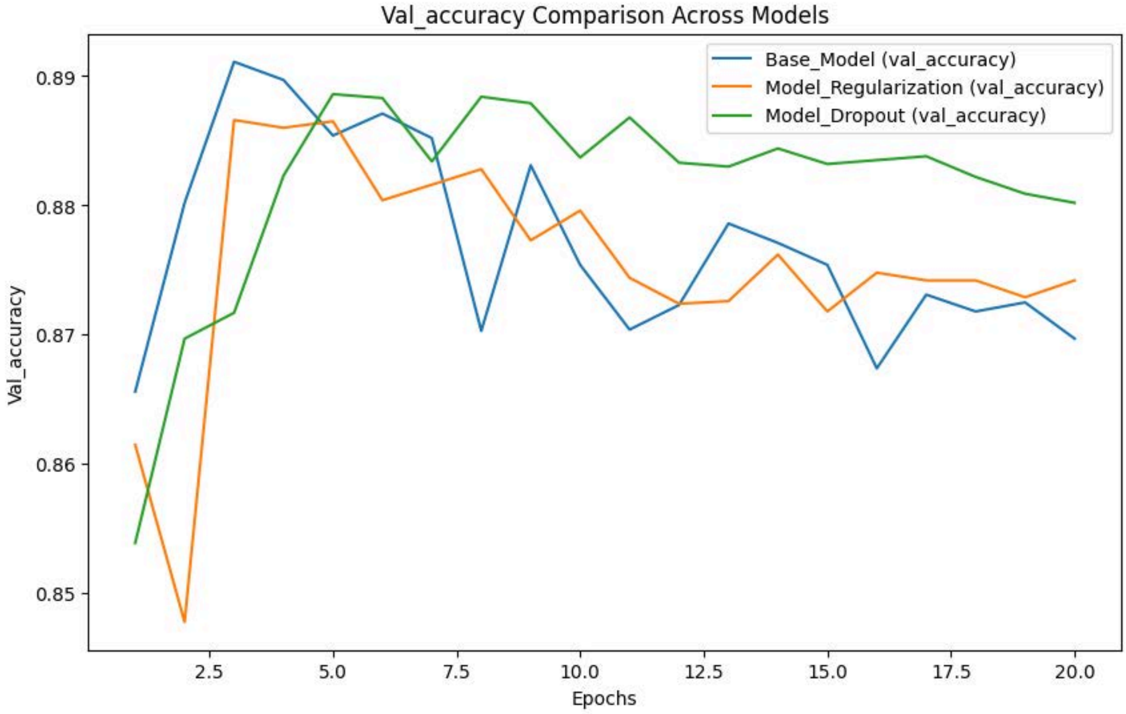


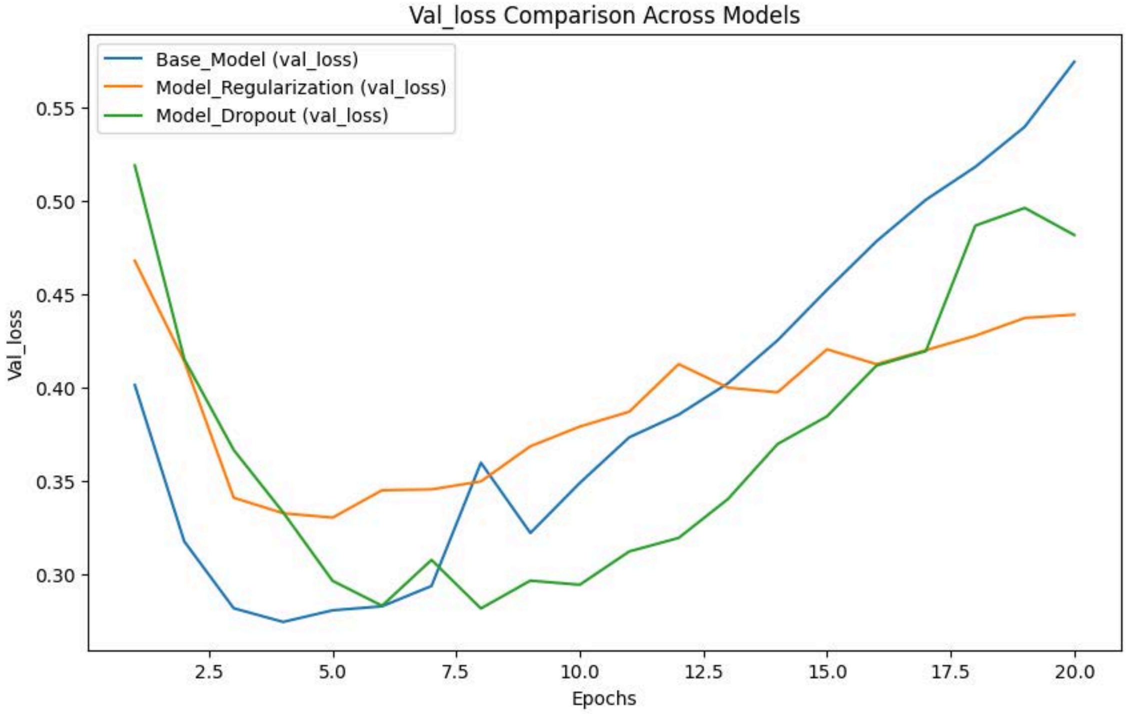


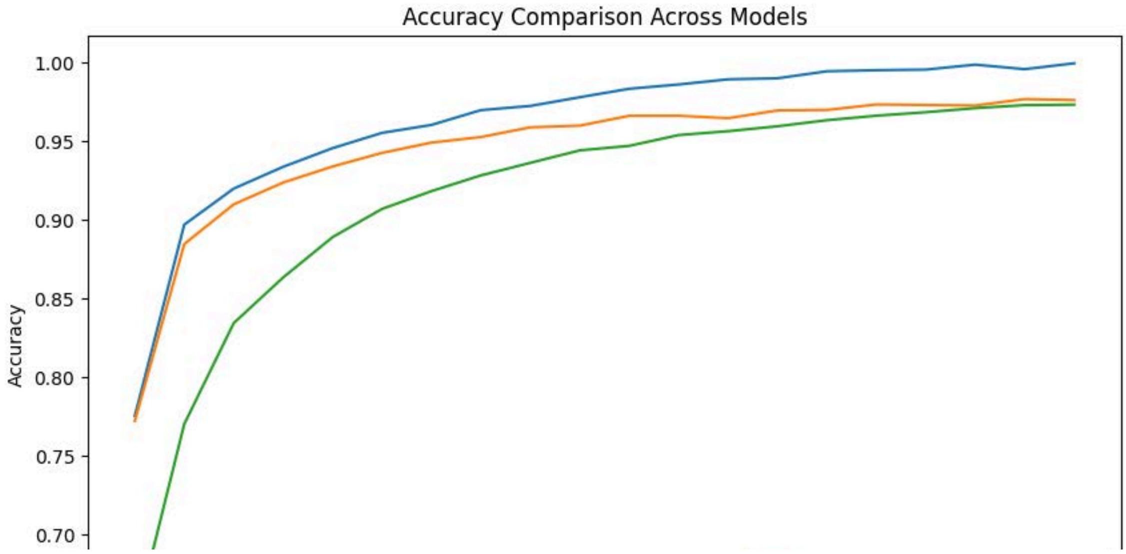
Question 5 - Comparison of L2 regularization, Dropout and Base model

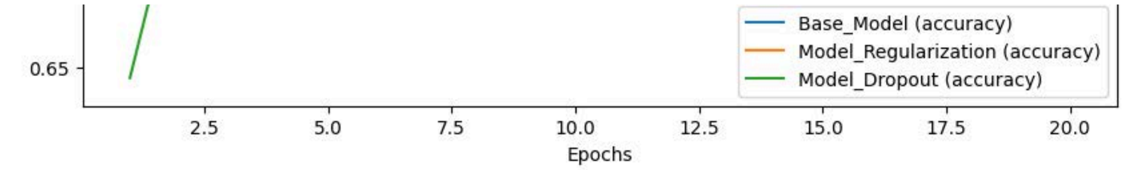
```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": Base_model,
    "Model_Regularization": Model_Reg_Tech,
    "Model_Dropout": Model_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')
```

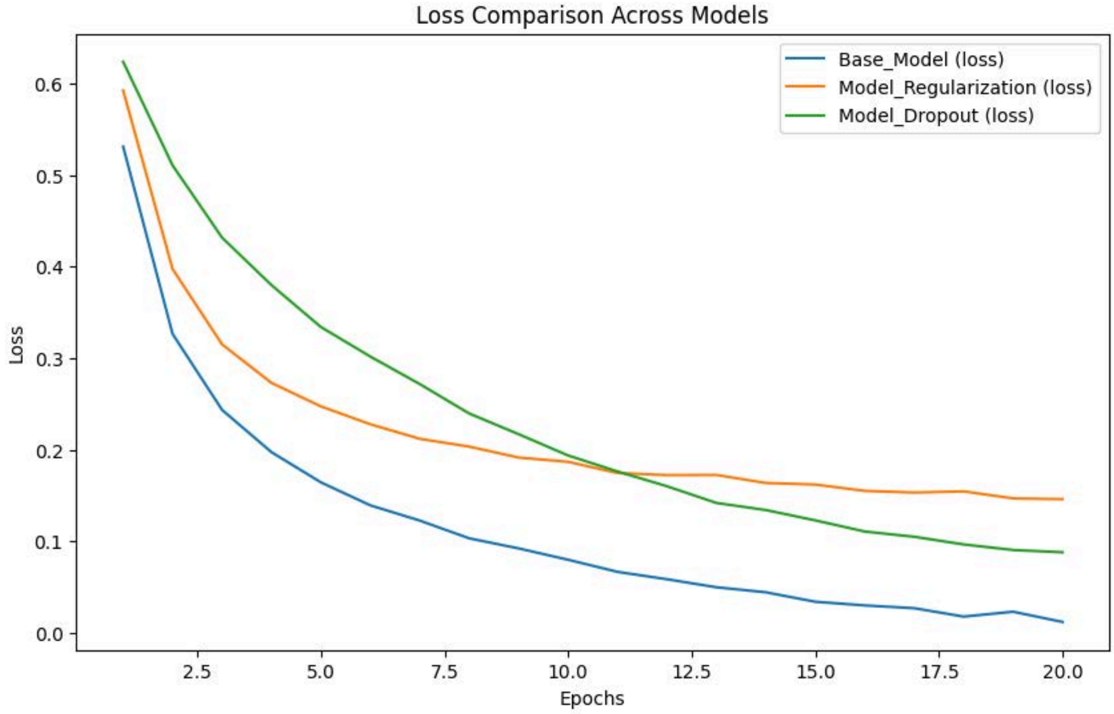
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_Regularization history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_Dropout history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```











Comparing all the models

```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": Base_model,
    "Model_1_Hidden_Layer": Model_1_Hidden_Layer,
    "Model_3_Hidden_Layer": Model_3_Hidden_Layer,
    "Model_32_Hidden_Units": Model_32_Hidden_Units,
    "Model_64_Hidden_Units": Model_64_Hidden_Units,
    "Model_MSE_Loss": Model_MSE_LOSS,
    "Model_TANH_Activation": Model_TANH_ACT,
    "Model_Regularization": Model_Reg_Tech,
    "Model_Dropout": Model_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_1_Hidden_Layer history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_3_Hidden_Layer history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_32_Hidden_Units history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_64_Hidden_Units history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_MSE_Loss history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```