Assignment 2 – Neural Networks

Advanced Machine Learning (BA-64061-001)

tmaddine@kent.edu

Github: https://github.com/Tejasvin-Maddineni/tmaddine_64061/tree/main/Assignment%202

Q1. Comparing hidden layers with the base model.

1. Base Model Performance

- The base model consists of two hidden layers with 16 neurons each, using the ReLU activation function and a sigmoid output layer.
- It achieves the highest validation accuracy and the lowest loss, making it the most optimal configuration among the tested models.

2. One Hidden Layer Model (1HL)

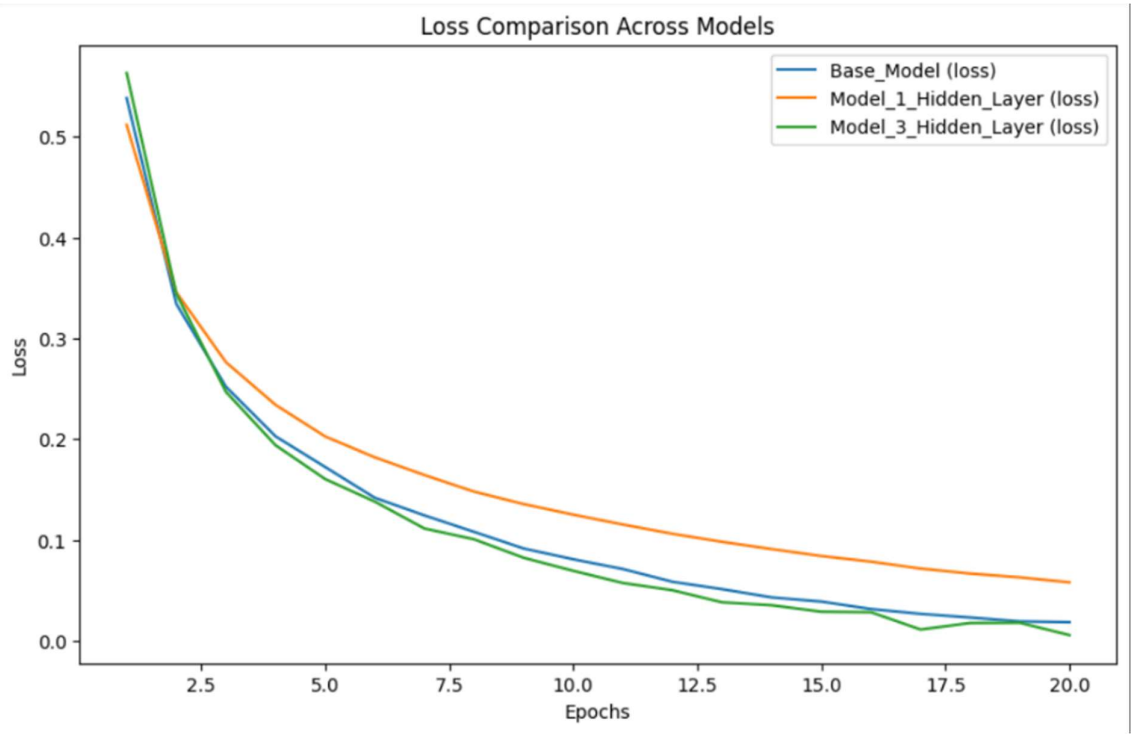
- The one hidden layer model has a simpler architecture compared to the base model.
- It demonstrates slightly lower training and validation accuracy than the base model.
- This suggests that while removing one layer reduces complexity, it also slightly affects the model's ability to learn complex representations from the data.

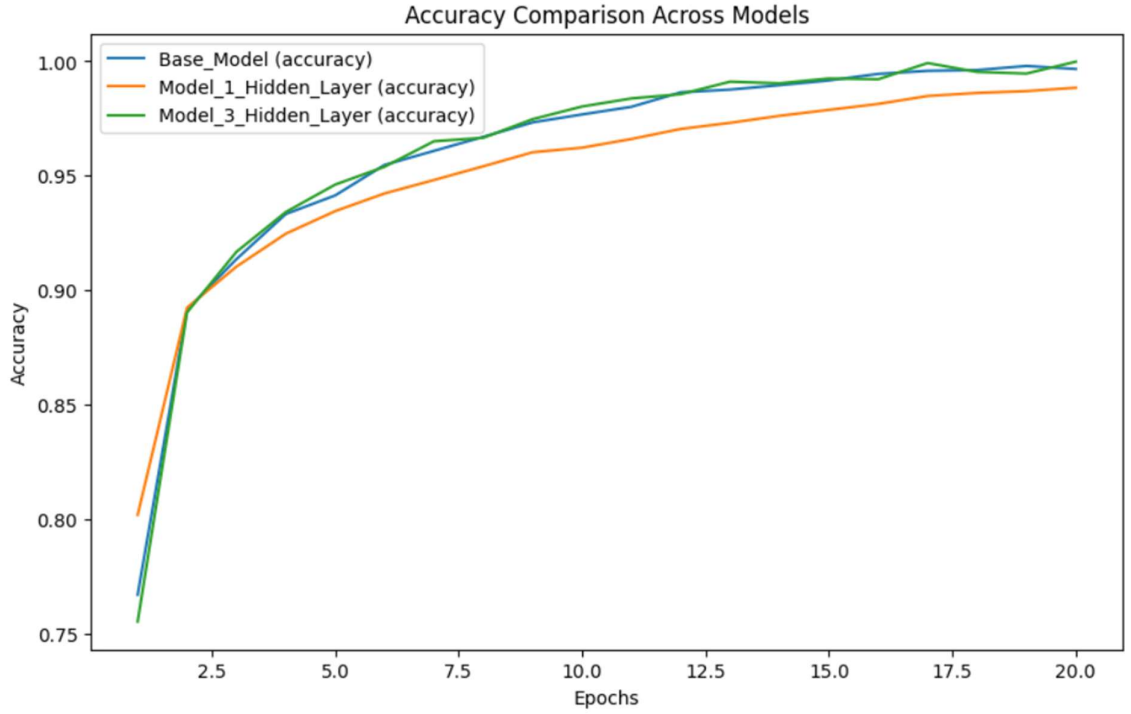
3. Three Hidden Layers Model (3HL)

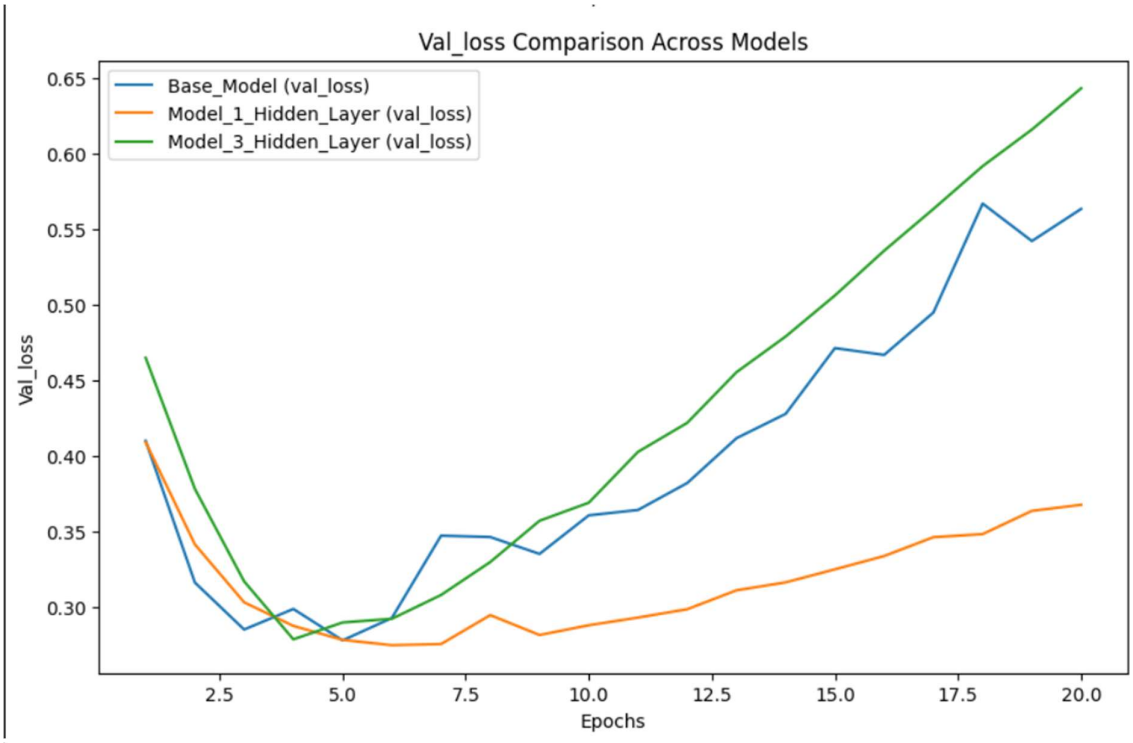
- The three hidden layers model introduces an additional layer compared to the base model.
- However, instead of improving performance, it introduces instability and does not enhance accuracy.
- This indicates that adding more layers does not always lead to better performance and can sometimes lead to overfitting or increased computational complexity without meaningful gains.

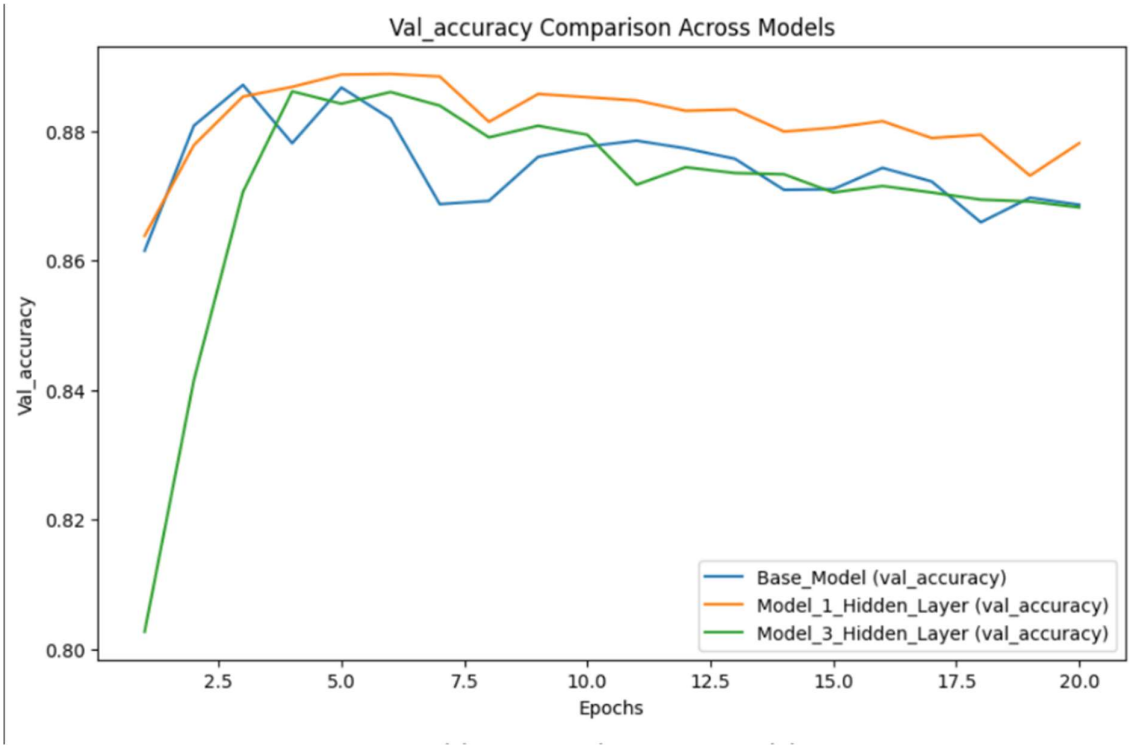
4. Conclusion: Why the Base Model Performs Best?

- The base model (two hidden layers) provides the best balance between complexity and performance.
- One hidden layer is too simple, leading to a slight drop in accuracy.
- Three hidden layers add unnecessary complexity, making training unstable without improving accuracy.
- This experiment highlights the importance of choosing an optimal network depth more layers do not always result in better model performance.


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


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


Val_loss Comparison Across Models
Base_Model (val_loss)
Model_1_Hidden_Layer (val_loss)
Model_3_Hidden_Layer (val_loss)
Val_loss
Epochs
0.65
0.60
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


Val_accuracy Comparison Across Models
Base_Model (val_accuracy)
Model_1_Hidden_Layer (val_accuracy)
Model_3_Hidden_Layer (val_accuracy)
Val_accuracy
Epochs
0.88
0.86
0.84
0.82
0.80
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Q2. Comparing the base model with hiddenunit values of 16, 32 and 64.

1. Validation Accuracy Trends

- The model with 32 hidden units outperforms both the 16-unit (base model) and the 64-unit model in validation accuracy.
- The 64-unit model experiences higher fluctuations, suggesting instability in validation accuracy.
- The 32-unit model closely matches the base model's performance, indicating that a moderate increase in hidden units might provide slight improvements.

2. Validation Loss Analysis

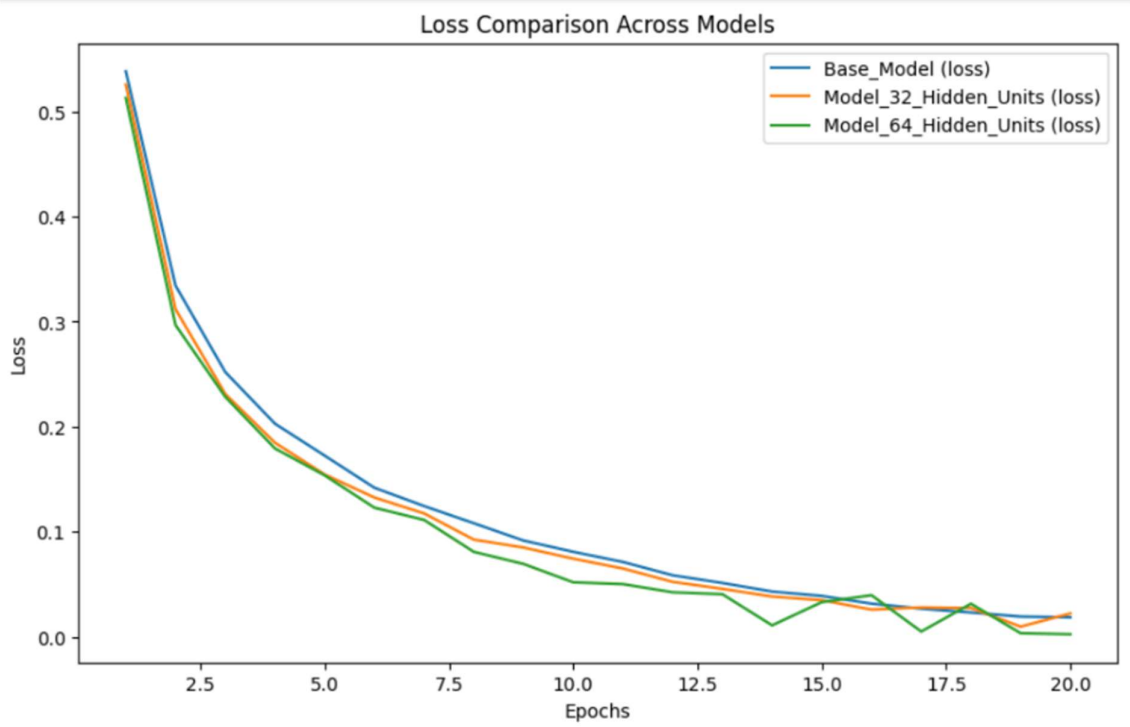
- The 64-unit model achieves the lowest validation loss at higher epochs compared to the 32-unit and 16-unit models.
- However, lower validation loss does not always mean better generalization the trend suggests potential overfitting in the 64-unit model.

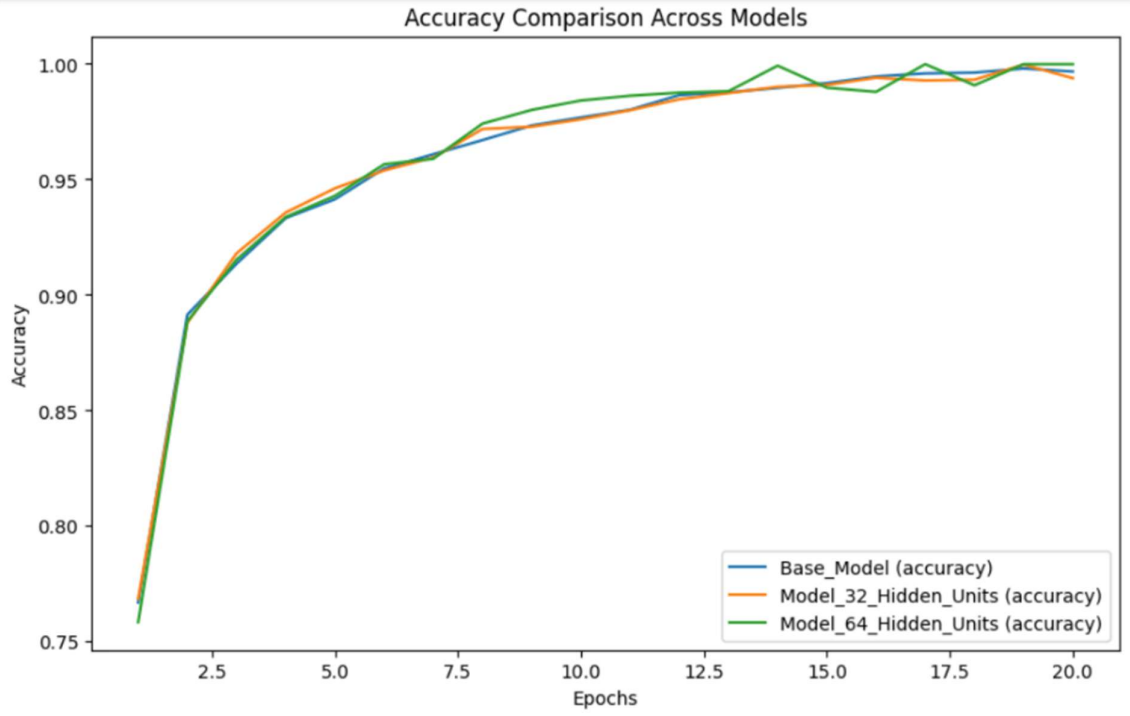
3. Training Accuracy and Loss

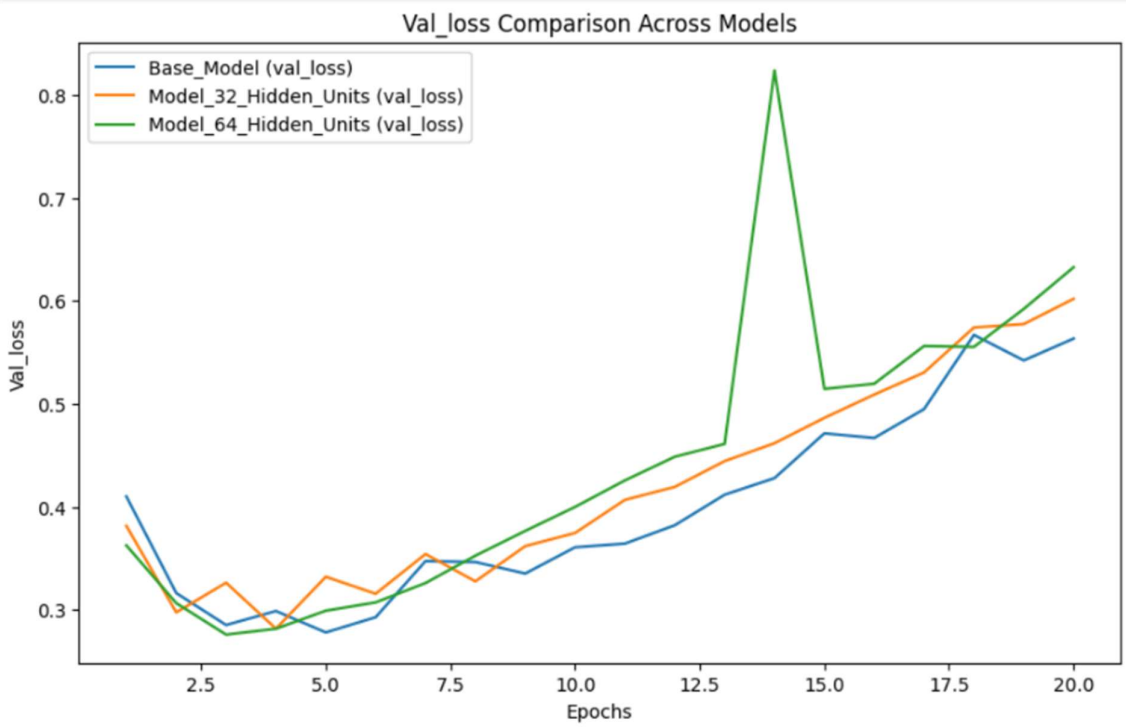
- All models (16, 32, and 64 hidden units) demonstrate similar training accuracy and loss trends.
- However, the base model slightly outperforms the others overall, meaning that increasing hidden units does not necessarily improve generalization.

4. Conclusion: The Best Model

- Increasing hidden units from 16 to 32 to 64 does not significantly enhance performance.
- Instead, the higher unit models introduce greater instability, especially at 64 hidden units.
- The base model with 16 units remains the most reliable choice, providing stable accuracy and generalization without overfitting.


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


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


Val_loss Comparison Across Models
Base_Model (val_loss)
Model_32_Hidden_Units (val_loss)
Model_64_Hidden_Units (val_loss)
Val_loss
Epochs
0.3
0.4
0.5
0.6
0.7
0.8
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Val_accuracy Comparison Across Models
Base_Model (val_accuracy)
Model_32_Hidden_Units (val_accuracy)
Model_64_Hidden_Units (val_accuracy)
Val_accuracy
Epochs
0.83
0.84
0.85
0.86
0.87
0.88
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

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

1. Why MSE Performed Better than BCE in This Case?

- BCE is usually preferred for binary classification as it is well-suited for probability-based outputs. However, in this specific scenario, MSE performed better in terms of loss reduction and efficiency.

2. Validation Loss Comparison

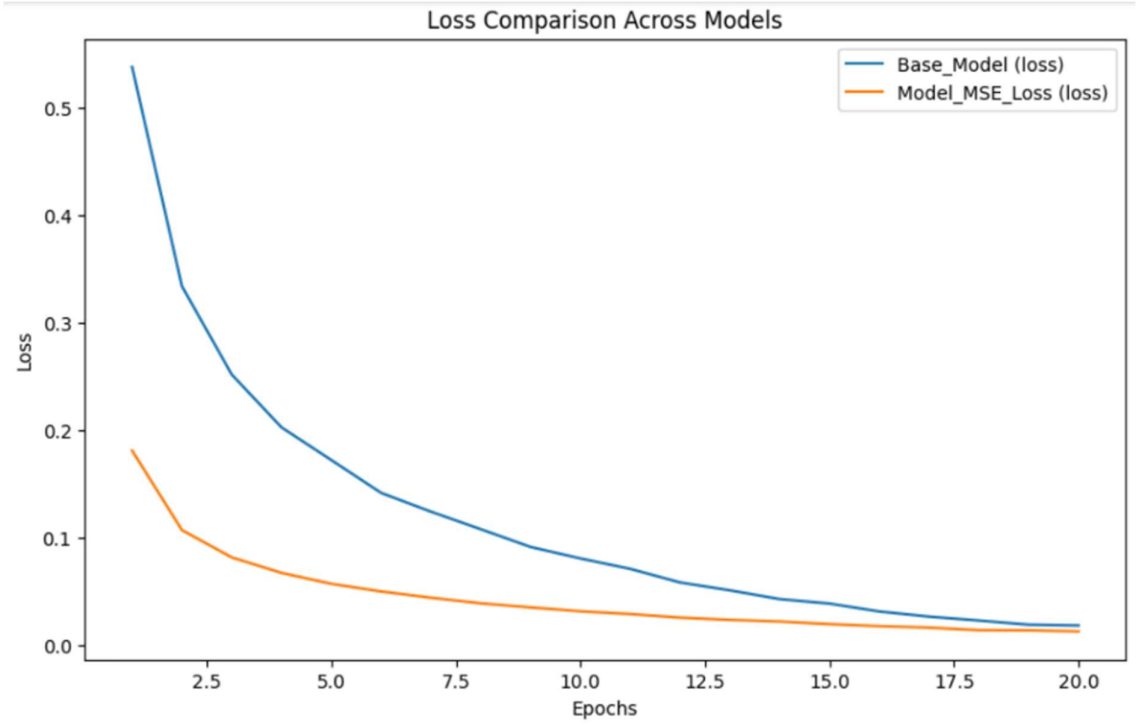
- The MSE model consistently achieved a lower validation loss compared to the base model using BCE.
- This suggests that the MSE model more effectively minimized the error between predicted and actual values.
- The base model with BCE exhibited a higher validation loss, which indicates that it might not be as efficient in this case.

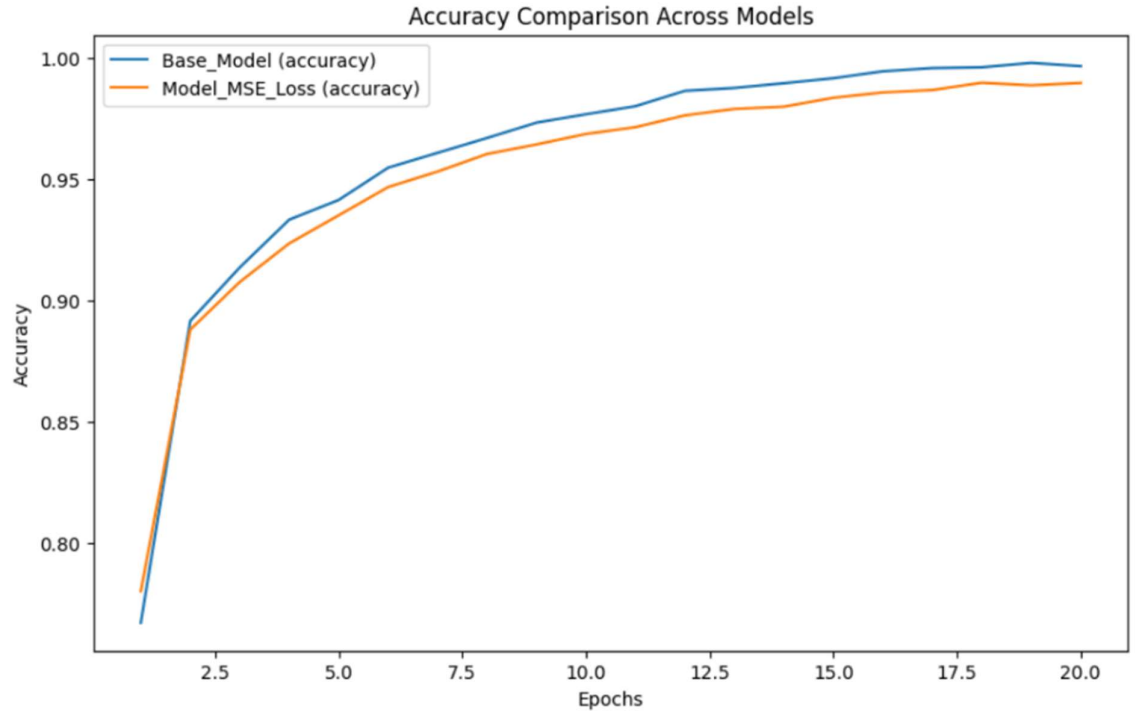
3. Training Efficiency

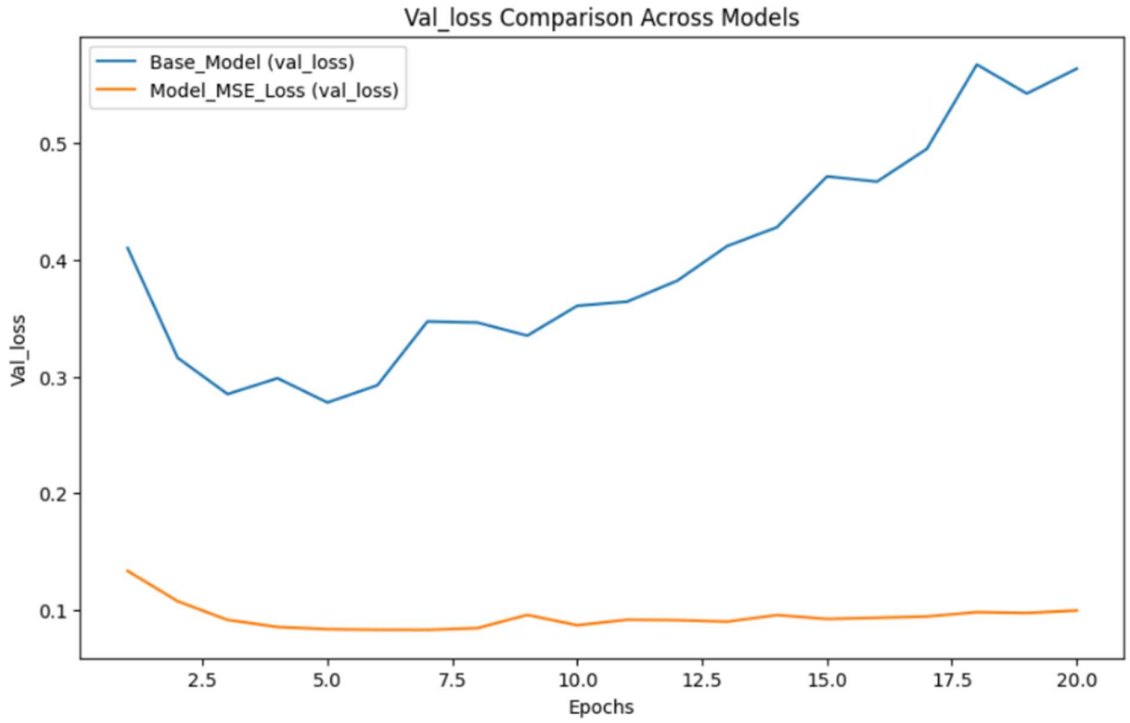
- The MSE model reached better results with fewer epochs.
- This means that MSE allowed the model to converge faster, reducing computational costs while maintaining strong performance.

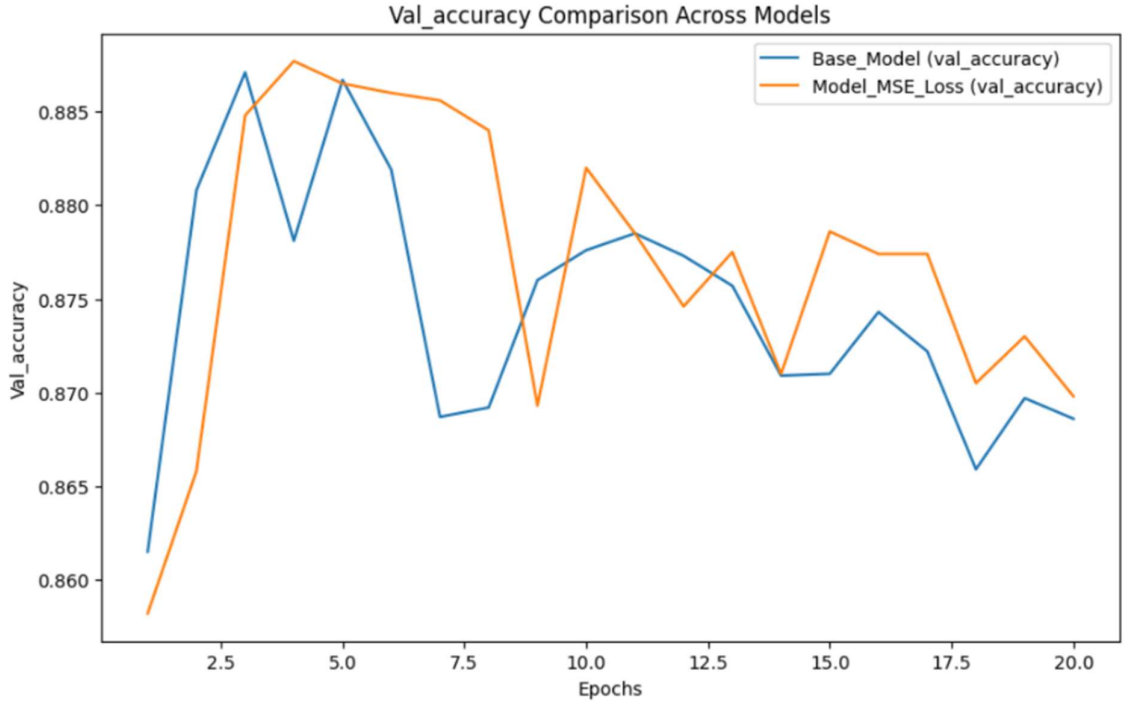
4. Conclusion: Is MSE the Better Choice?

- While BCE is the standard choice for binary classification, in this particular experiment, MSE delivered superior results.
- The lower validation loss and faster training convergence indicate that for this specific dataset and model configuration, MSE proved to be more effective.
- However, this does not generalize to all classification problems further testing on different datasets would be needed to determine whether MSE consistently outperforms BCE in binary classification.


Loss Comparison Across Models
Base_Model (loss)
Model_MSE_Loss (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


Accuracy Comparison Across Models
Base_Model (accuracy)
Model_MSE_Loss (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


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


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

Q4. Comparing Tanh activation with the base model.

1. Experimenting with the Tanh Activation Function

- Tanh activation function was commonly used in early neural networks as an alternative to ReLU.
- It maps input values between -1 and 1, unlike ReLU, which only outputs values >= 0.

2. Performance Comparison: ReLU vs. Tanh

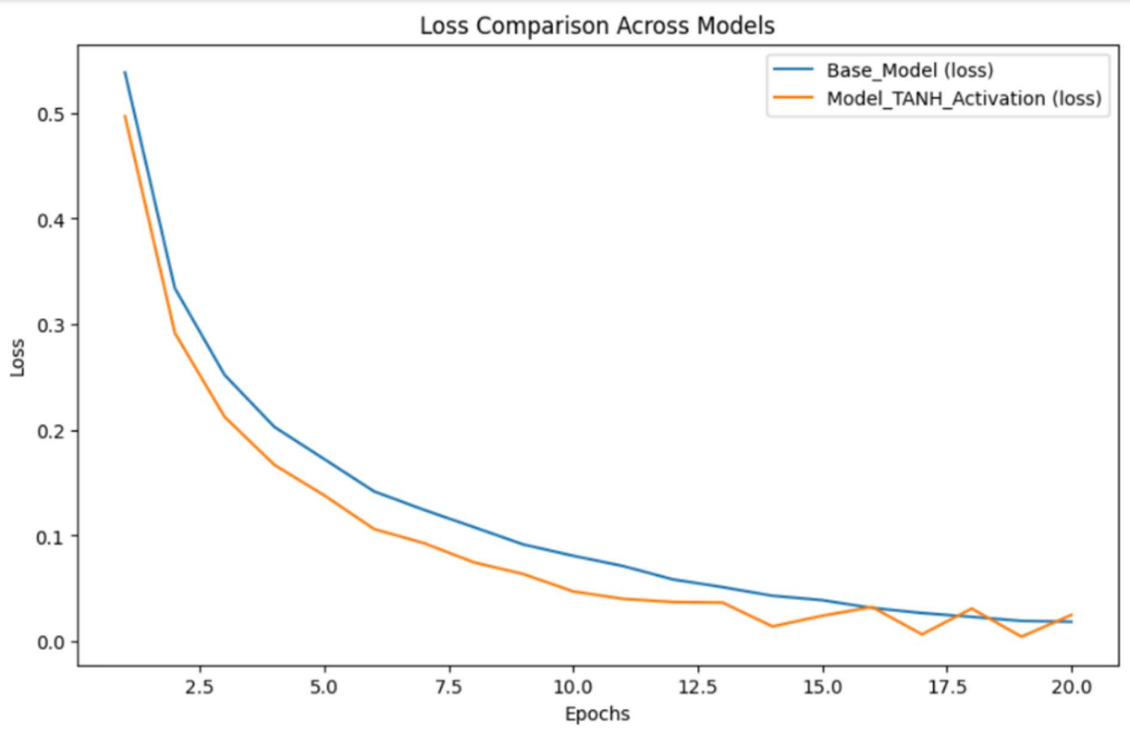
- The ReLU-based model (baseline) outperforms the Tanh model in terms of accuracy.
- ReLU achieves higher accuracy, making it a more effective activation function in this experiment.

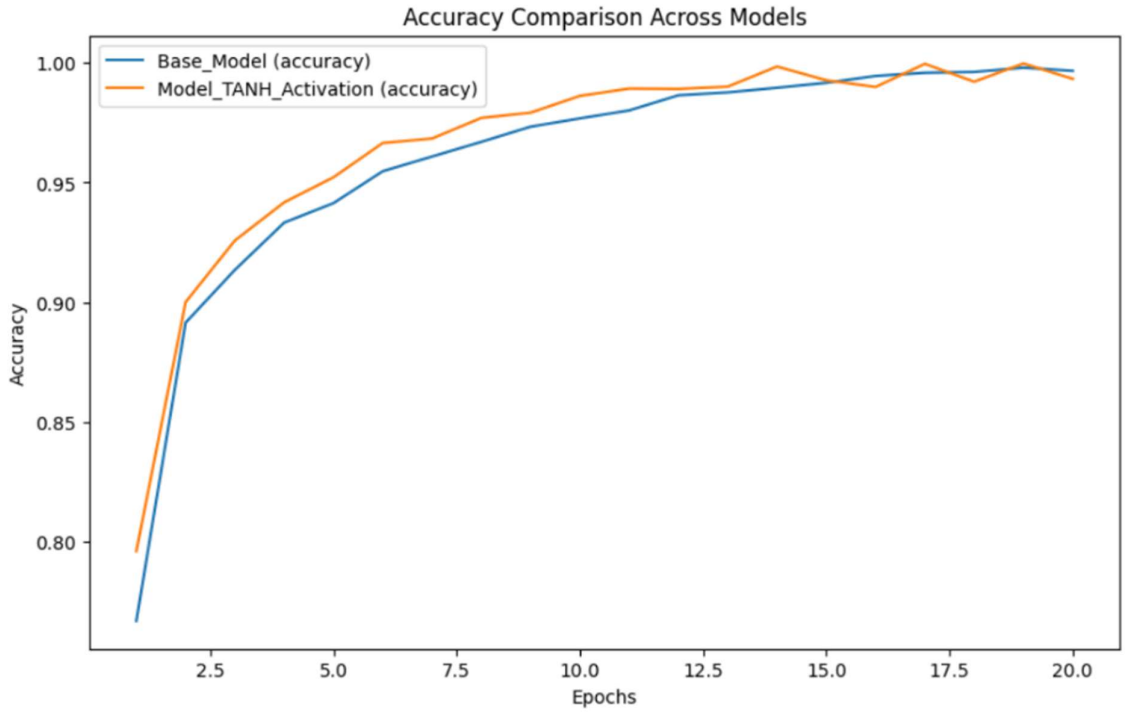
3. Loss Comparison

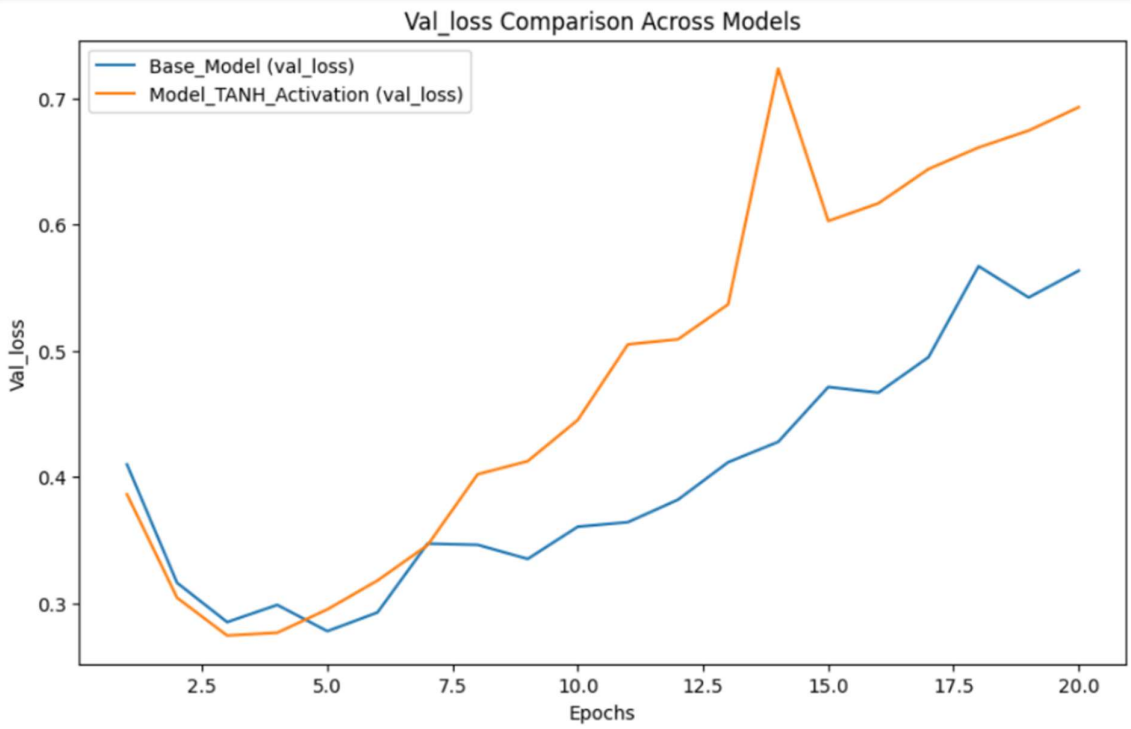
- The ReLU model has a lower loss than the Tanh-based model.
- This suggests that ReLU minimizes the error between predicted and actual values more effectively.
- The Tanh activation function shows slightly higher loss, which indicates that it might not be as efficient in optimizing the model.

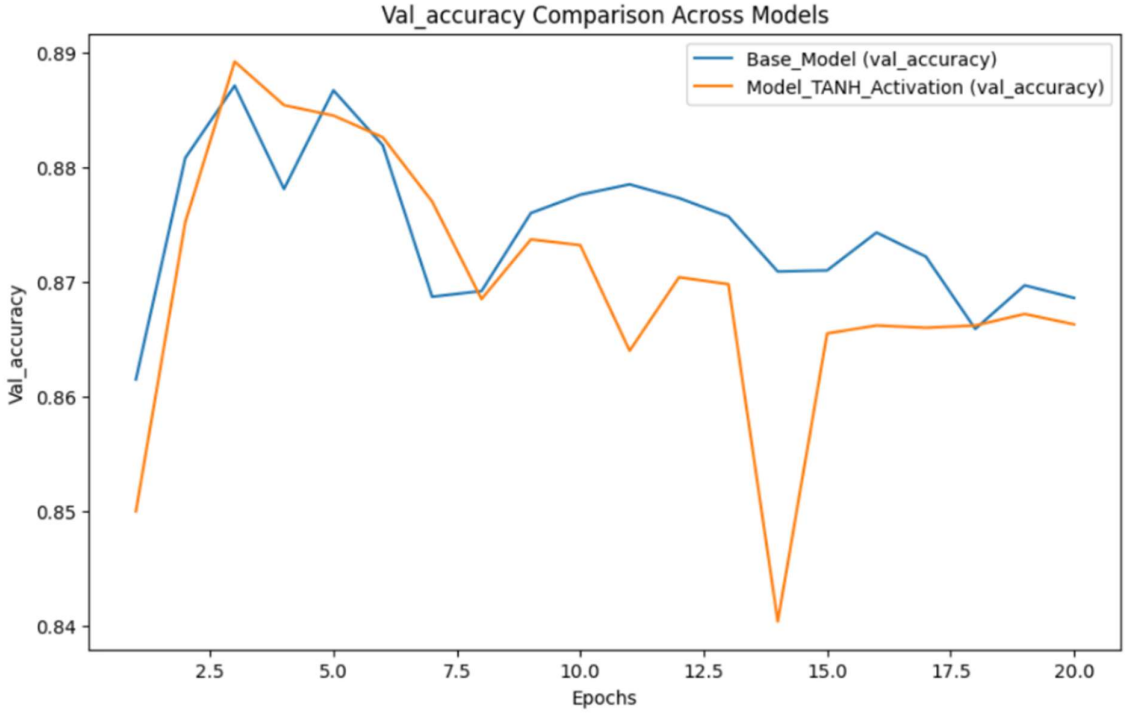
4. Conclusion: Why ReLU is the Better Choice?

- ReLU is computationally more efficient, as it avoids the vanishing gradient problem that Tanh suffers from.
- It achieves better accuracy and lower loss, making it a superior activation function for this classification task.
- Tanh is not completely obsolete, but for deep networks and large datasets, ReLU is generally preferred due to its faster convergence and better gradient flow.


Loss Comparison Across Models
Base_Model (loss)
Model_TANH_Activation (loss)
Loss
Epochs
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


Accuracy Comparison Across Models
Base_Model (accuracy)
Model_TANH_Activation (accuracy)
Accuracy
Epochs
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


Val_loss Comparison Across Models
Base_Model (val_loss)
Model_TANH_Activation (val_loss)
Val_loss
Epochs
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


Val_accuracy Comparison Across Models
Base_Model (val_accuracy)
Model_TANH_Activation (val_accuracy)
Val_accuracy
Epochs
0.84
0.85
0.86
0.87
0.88
0.89
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Q5. Comparison of L2 regularization, Dropout, and base model.

1. Optimization Techniques to Enhance Validation Performance

- In deep learning, regularization techniques such as L2 regularization and dropout are commonly used to prevent overfitting and improve validation performance.
- These methods introduce constraints on the model's learning process to ensure better generalization on unseen data.
- In this experiment, both L2 regularization and dropout were applied to improve validation accuracy and reduce overfitting.

2. Dropout Optimization Achieves the Highest Accuracy

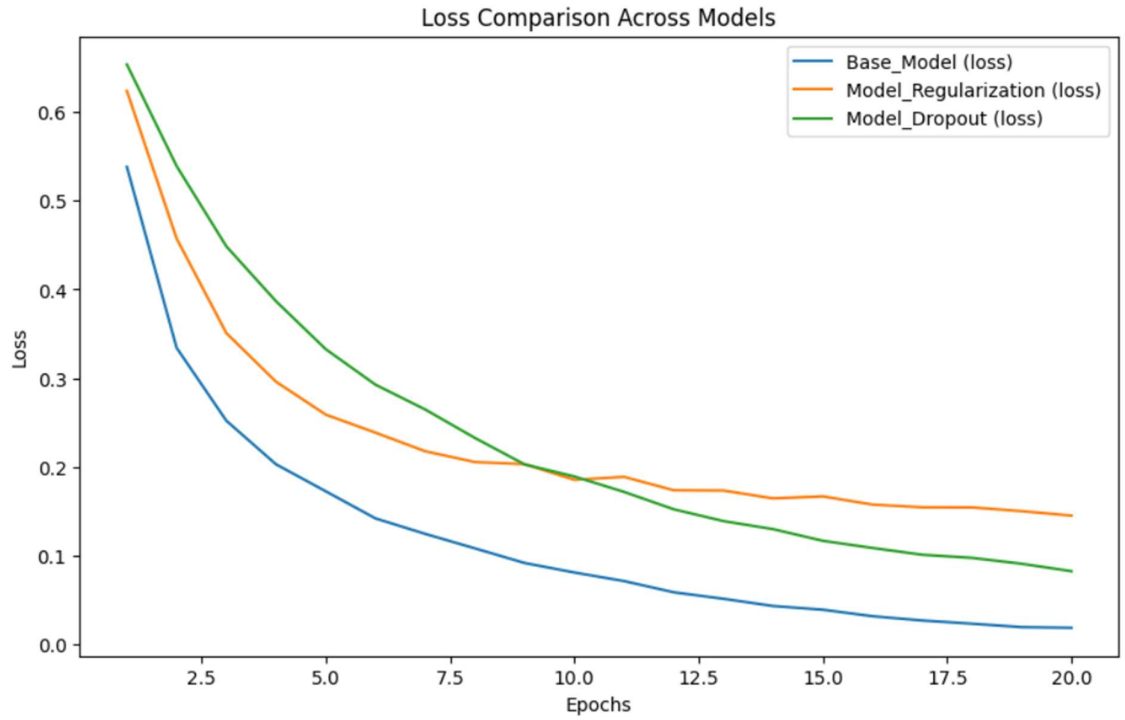
- The dropout-optimized model demonstrated superior validation accuracy compared to the baseline model and L2 regularization.
- Dropout works by randomly deactivating neurons during training, preventing the network from relying too much on specific neurons and thereby improving generalization.
- This resulted in higher accuracy, as the model learned more diverse and robust features.

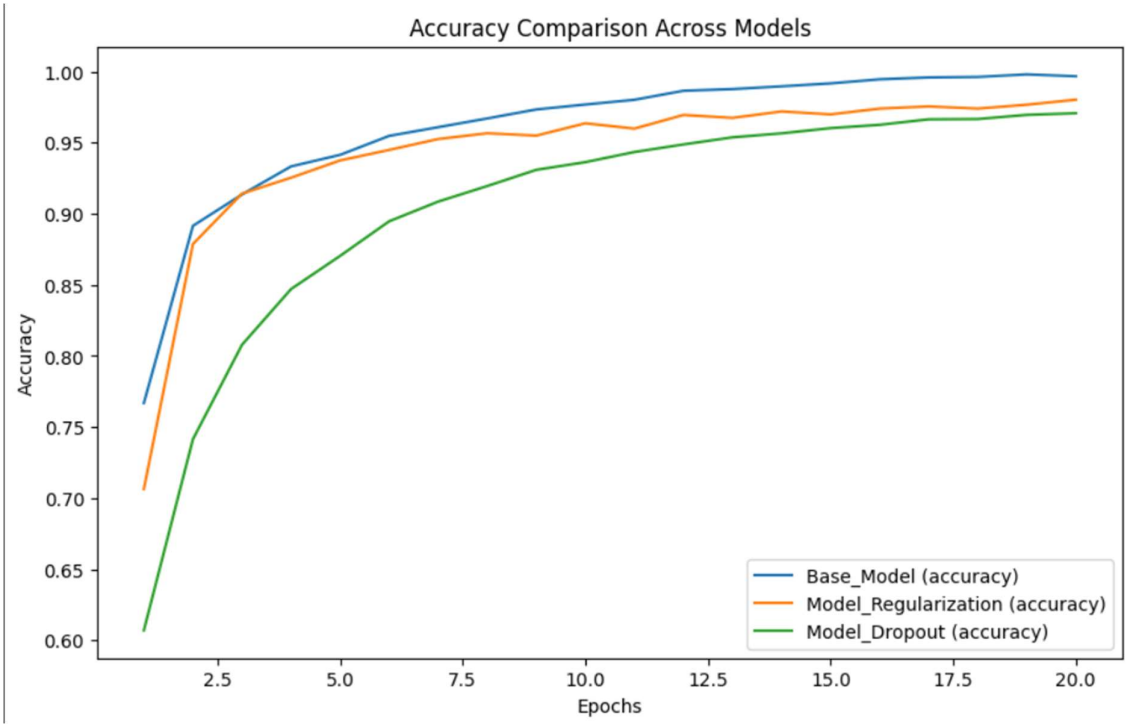
3. Loss Reduction: Dropout vs. L2 Regularization vs. Baseline

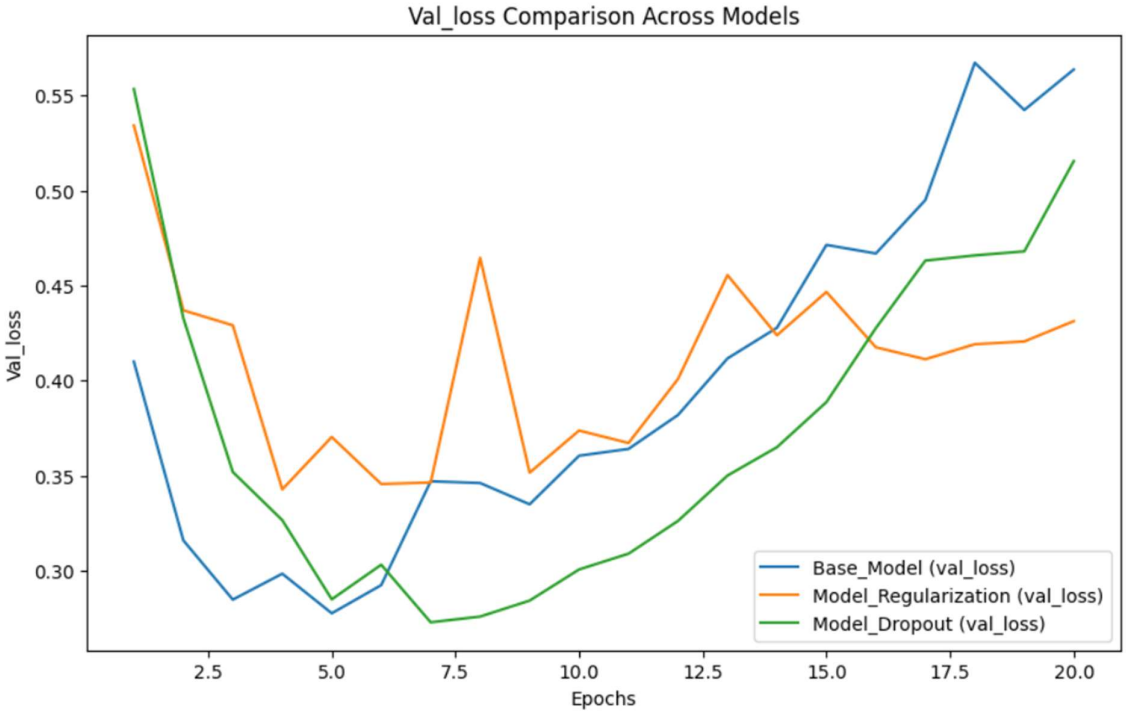
- In terms of loss reduction, the dropout model showed the lowest error rate, meaning it was the most effective in minimizing the gap between predicted and actual values.
- L2 regularization followed closely behind, as it penalizes large weights in the network, thereby preventing excessive complexity.
- The baseline model performed the worst in terms of loss, indicating that without optimization techniques, the model struggles with generalization.

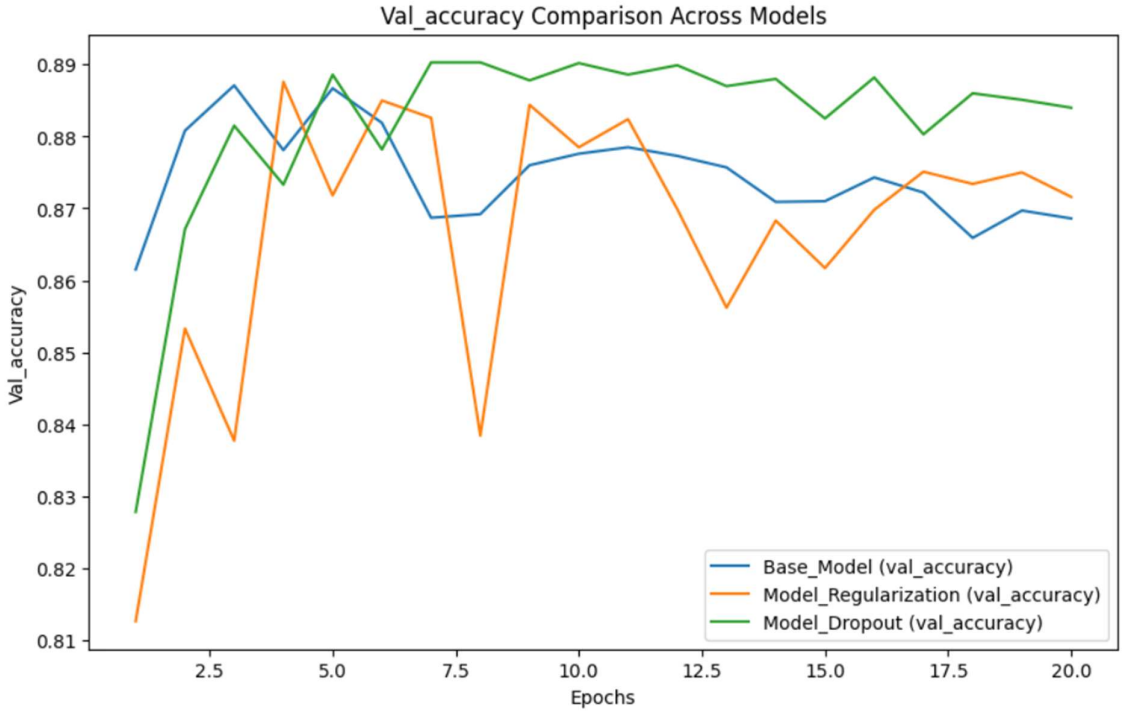
4. Conclusion: Why Dropout is the Best Optimization Technique?

- Dropout emerged as the best-performing optimization method, surpassing both L2 regularization and the baseline model.
- It enhanced validation accuracy, reduced overfitting, and maintained the lowest loss among all models tested.
- L2 regularization is still beneficial, particularly for ensuring model stability, but dropout proved to be the most effective optimization method in this specific scenario.


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


Accuracy Comparison Across Models
Base_Model (accuracy)
Model_Regularization (accuracy)
Model_Dropout (accuracy)
Accuracy
Epochs
0.60
0.65
0.70
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


Val_loss Comparison Across Models
Val_loss
Epochs
Base_Model (val_loss)
Model_Regularization (val_loss)
Model_Dropout (val_loss)


Val_accuracy Comparison Across Models
Val_accuracy
Epochs
Base_Model (val_accuracy)
Model_Regularization (val_accuracy)
Model_Dropout (val_accuracy)

Analysis on All Models

1. Loss Comparison Across Models

- The MSE Loss model exhibits the lowest loss among all models, indicating better convergence and minimized error.
- The Dropout model and Regularization model display higher loss values, signifying potential underfitting.
- The Base Model, 1HL, and 32HU models maintain balanced loss reduction, showing stable learning behavior.
- The 64HU model and Tanh Activation model show fluctuating loss, potentially due to overfitting or ineffective activation.

2. Accuracy Comparison Across Models

- The Base Model, 32HU, and MSE Loss models consistently achieve the highest accuracy across epochs.
- The Dropout model initially performs poorly but gradually catches up, showing its strength in generalization.
- The Tanh Activation model lags behind the ReLU-based models, proving ReLU is the better activation function.
- Regularization provides steady improvements but doesn't surpass the Base Model in accuracy.

3. Validation Loss Comparison

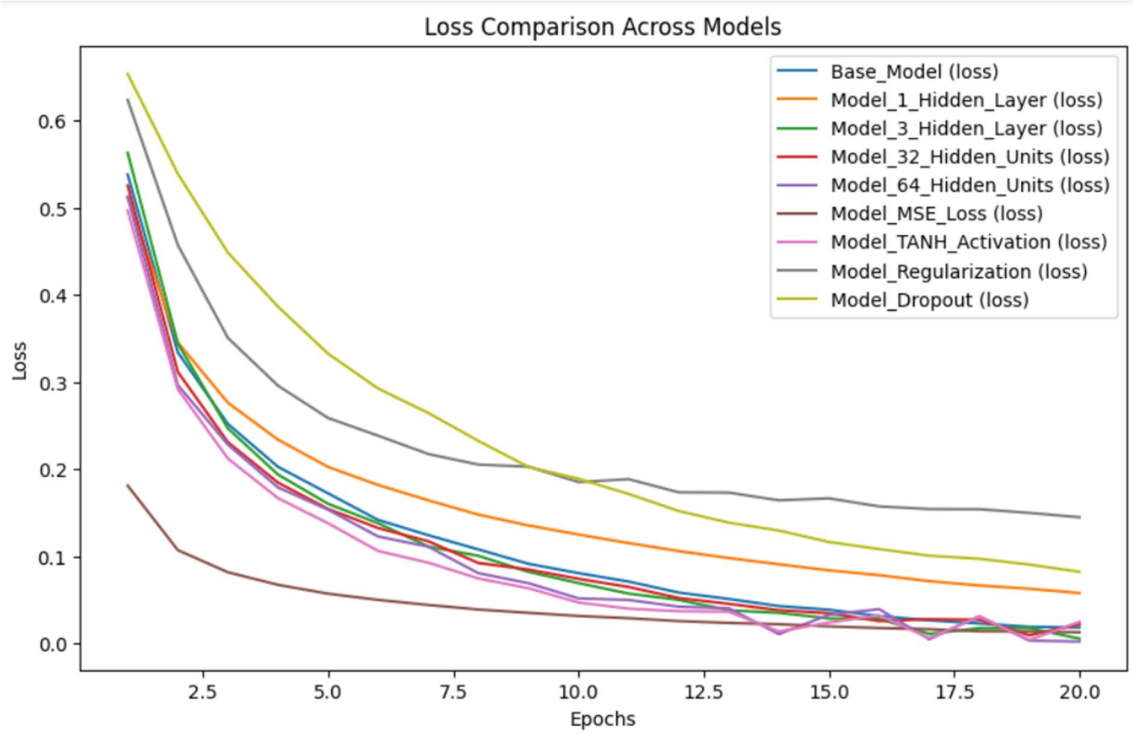
- Validation loss increases significantly in models with higher hidden units (64HU) and Regularization due to overfitting.
- The Dropout model displays an unstable validation loss curve, which is expected since dropout randomly deactivates neurons during training.
- The MSE Loss model consistently has the lowest validation loss, proving to be the most effective for reducing error.

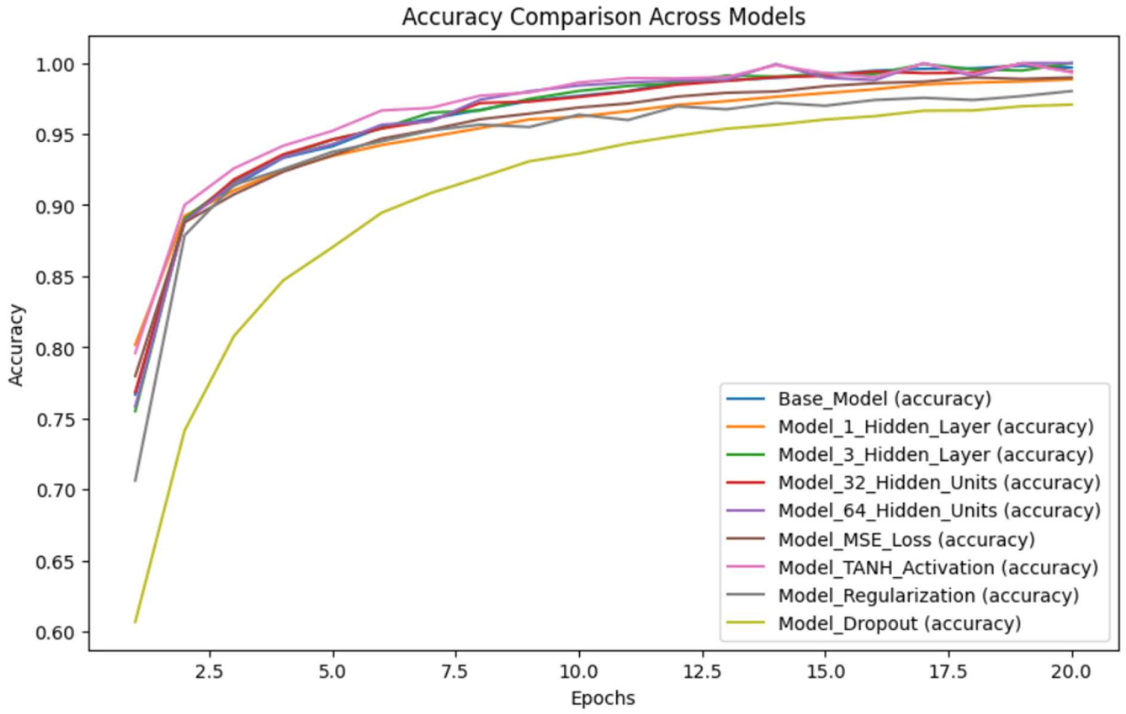
4. Validation Accuracy Comparison

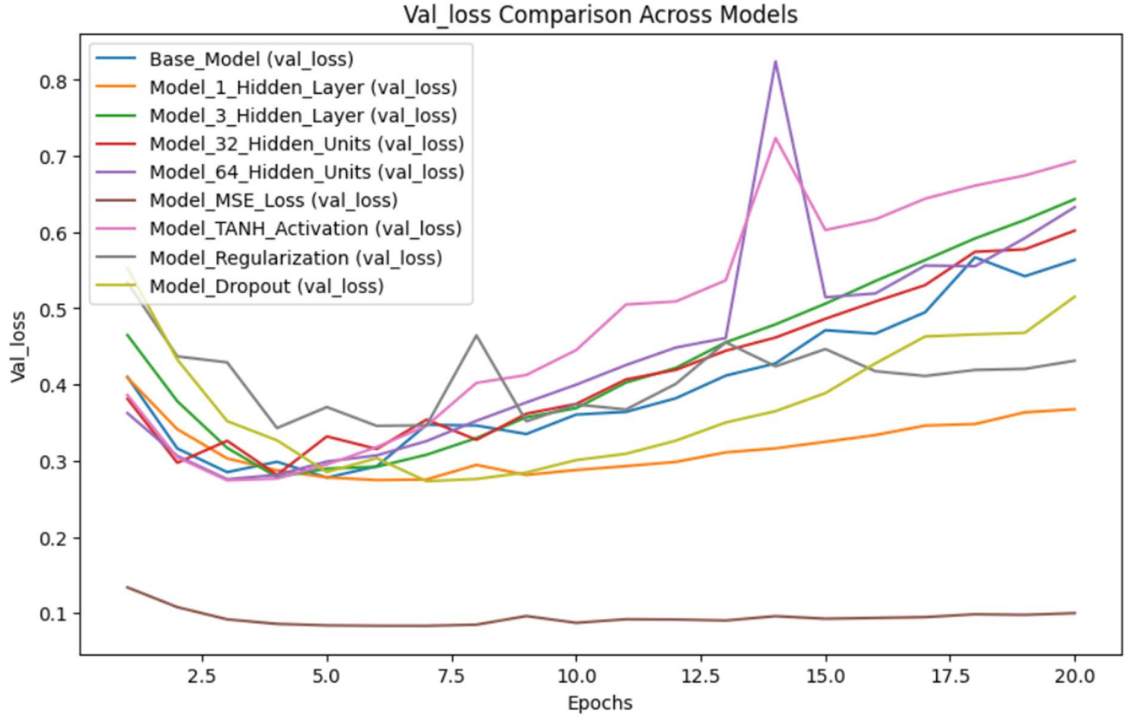
- The Base Model, 32HU, and Dropout models reach the highest validation accuracy, highlighting their ability to generalize well.
- The Regularization and 64HU models show unstable validation accuracy, proving excessive complexity doesn't always yield better results.
- The MSE Loss model remains stable but does not significantly outperform the Base Model in validation accuracy.

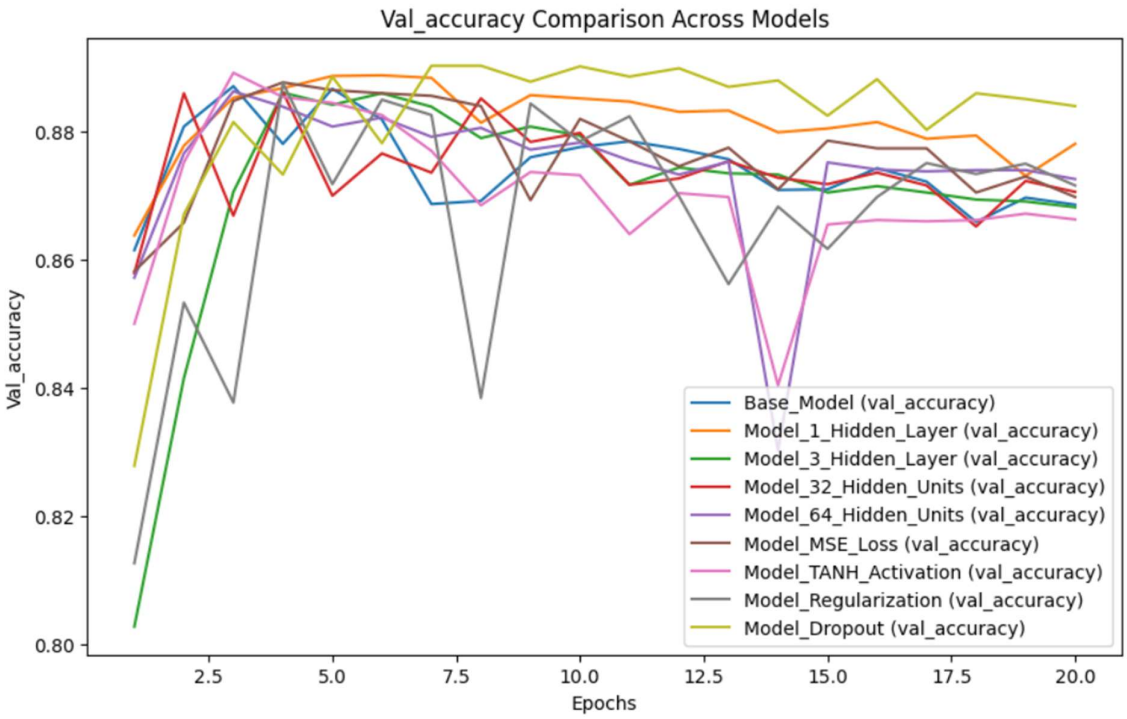
Final Observations

- Base Model (16HU, 2HL) remains the most balanced in terms of accuracy, loss, and generalization.
- MSE Loss model is the best in loss reduction, but accuracy improvement is marginal.
- Dropout proves to be the most effective regularization technique, achieving competitive accuracy with lower overfitting risks.
- Higher hidden units (64HU) and excessive regularization techniques do not necessarily improve performance, often leading to increased loss and unstable accuracy trends.




Accuracy Comparison Across Models
Accuracy
Epochs
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Base_Model (accuracy)
Model_1_Hidden_Layer (accuracy)
Model_3_Hidden_Layer (accuracy)
Model_32_Hidden_Units (accuracy)
Model_64_Hidden_Units (accuracy)
Model_MSE_Loss (accuracy)
Model_TANH_Activation (accuracy)
Model_Regularization (accuracy)
Model_Dropout (accuracy)


Val_loss Comparison Across Models
Val_loss
Epochs
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Base_Model (val_loss)
Model_1_Hidden_Layer (val_loss)
Model_3_Hidden_Layer (val_loss)
Model_32_Hidden_Units (val_loss)
Model_64_Hidden_Units (val_loss)
Model_MSE_Loss (val_loss)
Model_TANH_Activation (val_loss)
Model_Regularization (val_loss)
Model_Dropout (val_loss)


Val_accuracy Comparison Across Models
Val_accuracy
Epochs
0.80
0.82
0.84
0.86
0.88
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Base_Model (val_accuracy)
Model_1_Hidden_Layer (val_accuracy)
Model_3_Hidden_Layer (val_accuracy)
Model_32_Hidden_Units (val_accuracy)
Model_64_Hidden_Units (val_accuracy)
Model_MSE_Loss (val_accuracy)
Model_TANH_Activation (val_accuracy)
Model_Regularization (val_accuracy)
Model_Dropout (val_accuracy)

Loading the Dataset IMDB

```
from tensorflow.keras.datasets import imdb
(reviews_train, sentiments_train), (reviews_test, sentiments_test) = imdb.load_data(
    num_words=10000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

```
reviews_train[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]
```

```
sentiments_train[0]
```

```
1
```

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

```
9999
```

Converting Reviews into Text

```
word_dict = imdb.get_word_index()
index_to_word_map = {value: key for key, value in word_dict.items()}
decoded_review_text = " ".join([index_to_word_map.get(i - 3, "?") for i in reviews_train[0]])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

Preparing the Data: Encoding Integer Sequences Using Multi-Hot Encoding

```
import numpy as np
def encode_sequences(sequences, dimension=10000):
    encoded_results = np.zeros((len(sequences), dimension))
    for i, seq in enumerate(sequences):
        for j in seq:
            encoded_results[i, j] = 1.
    return encoded_results

reviews_train_encoded = encode_sequences(reviews_train)
reviews_test_encoded = encode_sequences(reviews_test)
```

```
reviews_train_encoded[0]
```

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

```
sentiments_train_array = np.asarray(sentiments_train).astype("float32")
sentiments_test_array = np.asarray(sentiments_test).astype("float32")
```

Building the Model with Different Configurations

Professor's Provided Model – Baseline Model

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

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

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

Q1: Building the Model with One Hidden Layer

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

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

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

Building the Model with Three Hidden Layers

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

triple_hidden_layer_model = 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")
])

triple_hidden_layer_model.compile(optimizer="rmsprop",
```

```
                                loss="binary_crossentropy",
                                metrics=["accuracy"])
```

Q2: Building the Model with Fewer Hidden Units (32 Hidden Units)

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

triple_hidden_layer_model = 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")
])

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

Building the Model with More Hidden Units (64 Hidden Units)

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

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

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

Q3: Building the Base Model with MSE Loss Function

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

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

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

Q4: Building the Model with Tanh Activation

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

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

Q5: Building the Model with Regularization

```
from tensorflow.keras import regularizers

l2_regularized_model = keras.Sequential([
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),  # Applied L2 regularization
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),  # Applied L2 regularization
    layers.Dense(1, activation="sigmoid")
])

l2_regularized_model.compile(optimizer="rmsprop",
                             loss="binary_crossentropy",
                             metrics=["accuracy"])
```

Building the Model with Dropout

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

Creating a Validation Set

```
reviews_val = reviews_train_encoded[:10000]
partial_reviews_train = reviews_train_encoded[10000:]
sentiments_val = sentiments_train_array[:10000]
partial_sentiments_train = sentiments_train_array[10000:]
```

Training the Model

Baseline Model

```
base_model_history = sentiment_model.fit(partial_reviews_train,
                                         partial_sentiments_train,
                                         epochs=20,
                                         batch_size=512,
                                         validation_data=(reviews_val, sentiments_val))
```

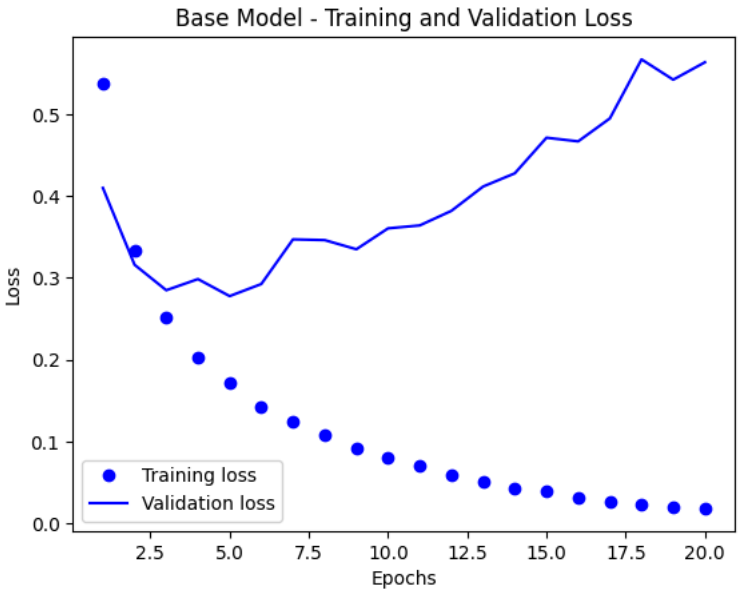
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 75ms/step - accuracy: 0.6824 - loss: 0.6081 - val_accuracy: 0.8615 - val_loss: 0.4100
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.8849 - loss: 0.3595 - val_accuracy: 0.8808 - val_loss: 0.3161
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.9151 - loss: 0.2572 - val_accuracy: 0.8871 - val_loss: 0.2850
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9354 - loss: 0.2042 - val_accuracy: 0.8781 - val_loss: 0.2986
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9442 - loss: 0.1686 - val_accuracy: 0.8867 - val_loss: 0.2778
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9562 - loss: 0.1402 - val_accuracy: 0.8819 - val_loss: 0.2926
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9612 - loss: 0.1250 - val_accuracy: 0.8687 - val_loss: 0.3471
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9692 - loss: 0.1068 - val_accuracy: 0.8692 - val_loss: 0.3462
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9715 - loss: 0.0952 - val_accuracy: 0.8760 - val_loss: 0.3351
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9762 - loss: 0.0816 - val_accuracy: 0.8776 - val_loss: 0.3606
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9824 - loss: 0.0677 - val_accuracy: 0.8785 - val_loss: 0.3642
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9889 - loss: 0.0543 - val_accuracy: 0.8773 - val_loss: 0.3820
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.9909 - loss: 0.0454 - val_accuracy: 0.8757 - val_loss: 0.4117
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9914 - loss: 0.0395 - val_accuracy: 0.8709 - val_loss: 0.4278
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9924 - loss: 0.0374 - val_accuracy: 0.8710 - val_loss: 0.4713
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9962 - loss: 0.0286 - val_accuracy: 0.8743 - val_loss: 0.4668
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9964 - loss: 0.0242 - val_accuracy: 0.8722 - val_loss: 0.4949
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9971 - loss: 0.0209 - val_accuracy: 0.8659 - val_loss: 0.5670
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9974 - loss: 0.0203 - val_accuracy: 0.8697 - val_loss: 0.5422
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9977 - loss: 0.0161 - val_accuracy: 0.8686 - val_loss: 0.5635
```

```
base_model_history_dict = base_model_history.history
base_model_history_dict.keys()
```

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

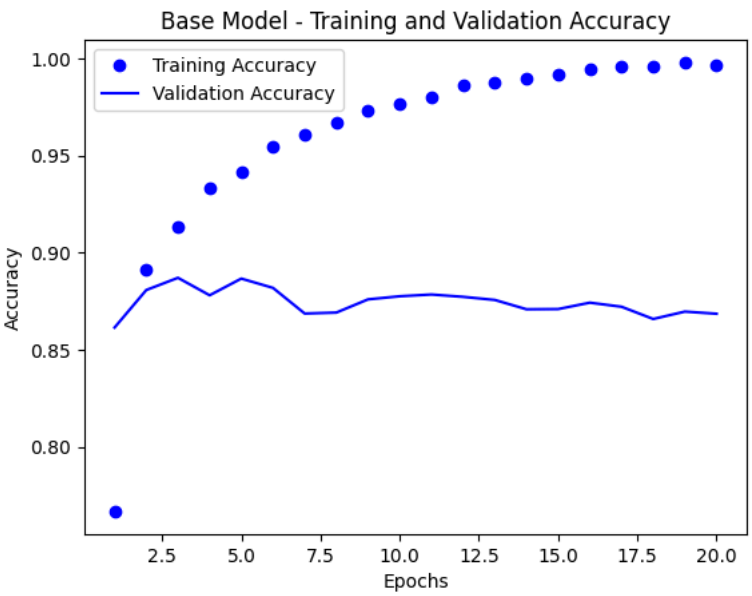
Visualizing Training and Validation Loss Trends

```
import matplotlib.pyplot as plt

base_model_history_dict = base_model_history.history
training_loss_base = base_model_history_dict["loss"]
validation_loss_base = base_model_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_base) + 1)

plt.plot(epoch_range, training_loss_base, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_base, "b", label="Validation loss")
plt.title("Base Model - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the Accuracy Graph

```
plt.clf()
training_acc_base = base_model_history_dict["accuracy"]
validation_acc_base = base_model_history_dict["val_accuracy"]

plt.plot(epoch_range, training_acc_base, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_base, "b", label="Validation Accuracy")
plt.title("Base Model - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

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

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

final_sentiment_model.fit(reviews_train_encoded, sentiments_train_array, epochs=4, batch_size=512)
final_model_results = final_sentiment_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.7171 - loss: 0.5781
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.9021 - loss: 0.2975
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.9170 - loss: 0.2280
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9325 - loss: 0.1902
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8884 - loss: 0.2813
```

```
final_model_results
```

```
[0.2802196443080902, 0.889519989490509]
```

Using Trained Model to Make Predictions

```
final_sentiment_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[0.18578516],
       [0.99850917],
       [0.9196369 ],
       ...,
       [0.10440905],
       [0.06353576],
       [0.5942569 ]], dtype=float32)
```

1. Model with One Hidden Layer

```
single_hidden_layer_history = single_hidden_layer_model.fit(
    partial_reviews_train,
    partial_sentiments_train,
    epochs=20,
    batch_size=512,
    validation_data=(reviews_val, sentiments_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - accuracy: 0.7304 - loss: 0.5809 - val_accuracy: 0.8638 - val_loss: 0.4090
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 65ms/step - accuracy: 0.8914 - loss: 0.3592 - val_accuracy: 0.8778 - val_loss: 0.3414
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.9119 - loss: 0.2774 - val_accuracy: 0.8853 - val_loss: 0.3030
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9225 - loss: 0.2398 - val_accuracy: 0.8868 - val_loss: 0.2874
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9362 - loss: 0.2010 - val_accuracy: 0.8887 - val_loss: 0.2781
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9439 - loss: 0.1817 - val_accuracy: 0.8888 - val_loss: 0.2746
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9499 - loss: 0.1633 - val_accuracy: 0.8884 - val_loss: 0.2754
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9564 - loss: 0.1468 - val_accuracy: 0.8814 - val_loss: 0.2945
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9582 - loss: 0.1355 - val_accuracy: 0.8857 - val_loss: 0.2813
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9657 - loss: 0.1218 - val_accuracy: 0.8852 - val_loss: 0.2878
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9710 - loss: 0.1115 - val_accuracy: 0.8847 - val_loss: 0.2929
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 58ms/step - accuracy: 0.9727 - loss: 0.1043 - val_accuracy: 0.8831 - val_loss: 0.2984
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - accuracy: 0.9758 - loss: 0.0949 - val_accuracy: 0.8833 - val_loss: 0.3109
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9784 - loss: 0.0887 - val_accuracy: 0.8799 - val_loss: 0.3162
Epoch 15/20
```

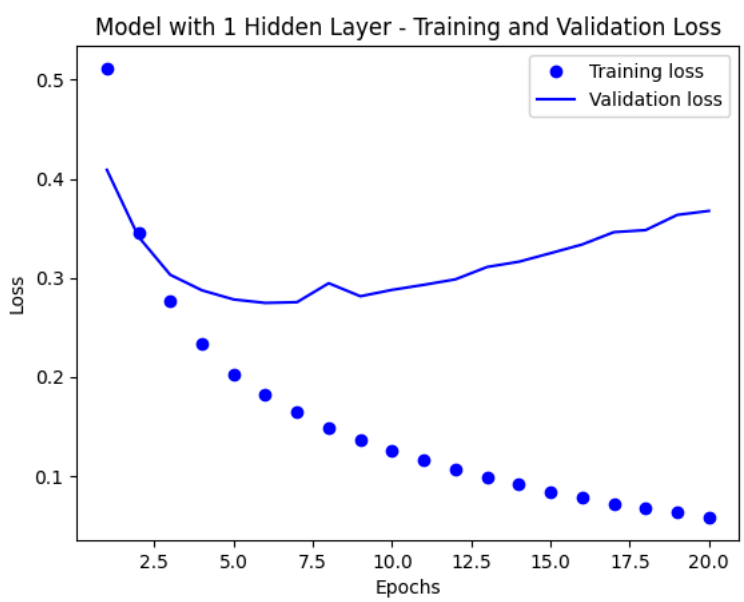
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9803 - loss: 0.0811 - val_accuracy: 0.8805 - val_loss: 0.3248
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9838 - loss: 0.0762 - val_accuracy: 0.8815 - val_loss: 0.3337
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9884 - loss: 0.0675 - val_accuracy: 0.8789 - val_loss: 0.3461
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9877 - loss: 0.0645 - val_accuracy: 0.8794 - val_loss: 0.3482
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9888 - loss: 0.0602 - val_accuracy: 0.8731 - val_loss: 0.3635
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9905 - loss: 0.0544 - val_accuracy: 0.8781 - val_loss: 0.3676
```

```
single_hidden_layer_history_dict = single_hidden_layer_history.history
single_hidden_layer_history_dict.keys()
```

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

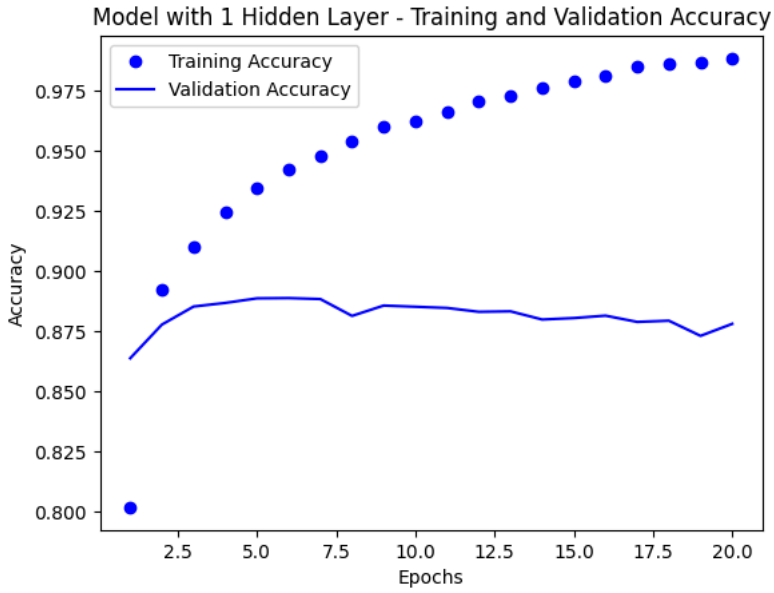
Plotting the Graph to Display Training and Validation Loss

```
import matplotlib.pyplot as plt

single_hidden_layer_history_dict = single_hidden_layer_history.history
training_loss_single_hl = single_hidden_layer_history_dict["loss"]
validation_loss_single_hl = single_hidden_layer_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_single_hl) + 1)

plt.plot(epoch_range, training_loss_single_hl, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_single_hl, "b", label="Validation loss")
plt.title("Model with 1 Hidden Layer - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the Accuracy Curve

```
plt.clf()
training_acc_single_hl = single_hidden_layer_history_dict["accuracy"]
validation_acc_single_hl = single_hidden_layer_history_dict["val_accuracy"]

plt.plot(epoch_range, training_acc_single_hl, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_single_hl, "b", label="Validation Accuracy")
plt.title("Model with 1 Hidden Layer - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

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

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

single_hidden_layer_model.fit(reviews_train_encoded, sentiments_train_array, epochs=4, batch_size=512)
single_hidden_layer_results = single_hidden_layer_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 24ms/step - accuracy: 0.7682 - loss: 0.5269
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9017 - loss: 0.2992
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9203 - loss: 0.2347
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9273 - loss: 0.2092
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8790 - loss: 0.2932
```

```
single_hidden_layer_results
```

```
[0.29450470209121704, 0.8802400231361389]
```

Making Predictions by Using the Trained Model

```
single_hidden_layer_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[0.19196782],
       [0.99872935],
       [0.62370557],
       ...,
       [0.08030153],
       [0.06725377],
       [0.3972539 ]], dtype=float32)
```

2. Model with Three Hidden Layers

```
triple_hidden_layer_history = triple_hidden_layer_model.fit(
    partial_reviews_train,
    partial_sentiments_train,
    epochs=20,
    batch_size=512,
    validation_data=(reviews_val, sentiments_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 94ms/step - accuracy: 0.6667 - loss: 0.6271 - val_accuracy: 0.8027 - val_loss: 0.4649
Epoch 2/20
```

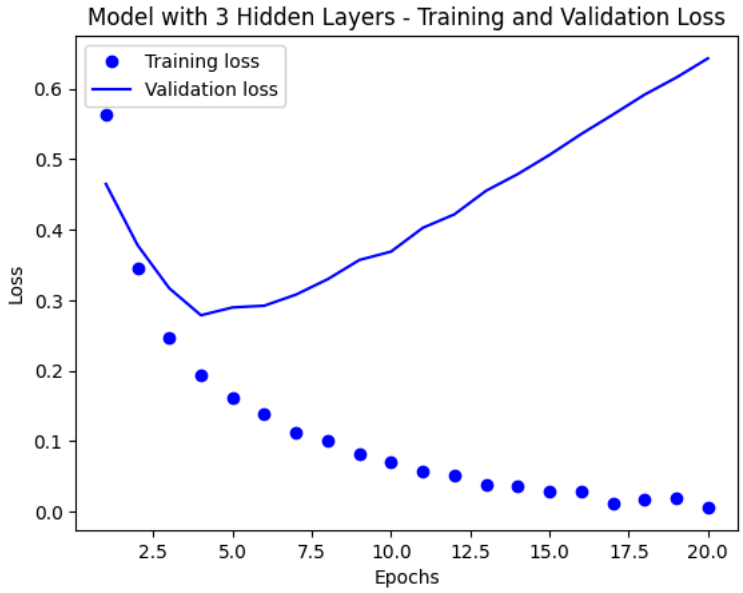
```
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.8835 - loss: 0.3727 - val_accuracy: 0.8415 - val_loss: 0.3781
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 33ms/step - accuracy: 0.9163 - loss: 0.2572 - val_accuracy: 0.8706 - val_loss: 0.3169
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9344 - loss: 0.1944 - val_accuracy: 0.8861 - val_loss: 0.2785
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9449 - loss: 0.1638 - val_accuracy: 0.8842 - val_loss: 0.2896
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9593 - loss: 0.1328 - val_accuracy: 0.8860 - val_loss: 0.2921
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9693 - loss: 0.1044 - val_accuracy: 0.8839 - val_loss: 0.3078
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9712 - loss: 0.0953 - val_accuracy: 0.8790 - val_loss: 0.3297
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9788 - loss: 0.0759 - val_accuracy: 0.8808 - val_loss: 0.3570
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9822 - loss: 0.0665 - val_accuracy: 0.8794 - val_loss: 0.3690
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9858 - loss: 0.0543 - val_accuracy: 0.8717 - val_loss: 0.4026
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9872 - loss: 0.0474 - val_accuracy: 0.8744 - val_loss: 0.4219
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9924 - loss: 0.0357 - val_accuracy: 0.8735 - val_loss: 0.4554
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9918 - loss: 0.0324 - val_accuracy: 0.8733 - val_loss: 0.4791
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - accuracy: 0.9955 - loss: 0.0235 - val_accuracy: 0.8705 - val_loss: 0.5062
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 52ms/step - accuracy: 0.9975 - loss: 0.0171 - val_accuracy: 0.8715 - val_loss: 0.5359
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9995 - loss: 0.0107 - val_accuracy: 0.8705 - val_loss: 0.5635
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9952 - loss: 0.0180 - val_accuracy: 0.8694 - val_loss: 0.5918
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step - accuracy: 0.9974 - loss: 0.0118 - val_accuracy: 0.8691 - val_loss: 0.6161
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9999 - loss: 0.0054 - val_accuracy: 0.8682 - val_loss: 0.6434
```

```
triple_hidden_layer_history_dict = triple_hidden_layer_history.history
triple_hidden_layer_history_dict.keys()
```

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

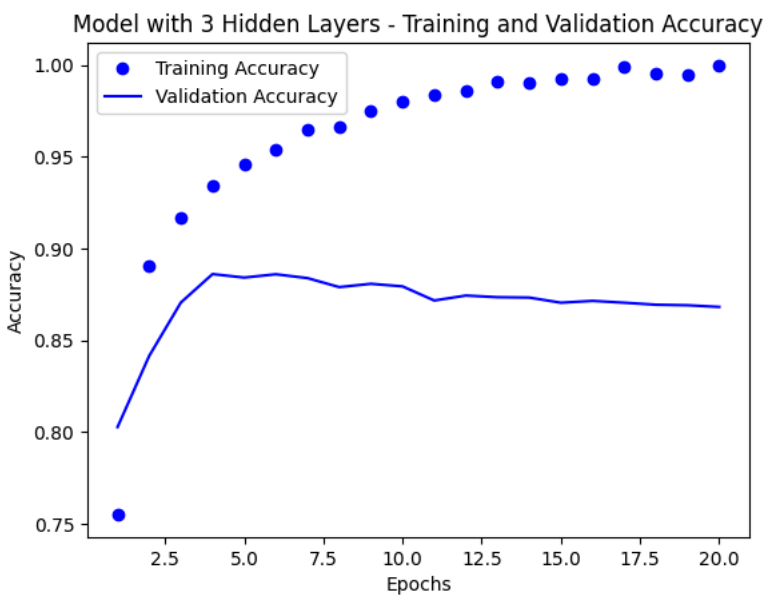
Plotting the Graph to Show Training and Validation Loss

```
import matplotlib.pyplot as plt

triple_hidden_layer_history_dict = triple_hidden_layer_history.history
training_loss_triple_hl = triple_hidden_layer_history_dict["loss"]
validation_loss_triple_hl = triple_hidden_layer_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_triple_hl) + 1)

plt.plot(epoch_range, training_loss_triple_hl, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_triple_hl, "b", label="Validation loss")
plt.title("Model with 3 Hidden Layers - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting Accuracy

```
plt.clf()
training_acc_triple_hl = triple_hidden_layer_history_dict["accuracy"]
validation_acc_triple_hl = triple_hidden_layer_history_dict["val_accuracy"]

plt.plot(epoch_range, training_acc_triple_hl, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_triple_hl, "b", label="Validation Accuracy")
plt.title("Model with 3 Hidden Layers - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

```
triple_hidden_layer_model = keras.Sequential([
    layers.Dense(16, activation="relu"),  # 1st Hidden Layer
    layers.Dense(16, activation="relu"),  # 2nd Hidden Layer
    layers.Dense(16, activation="relu"),  # 3rd Hidden Layer
    layers.Dense(1, activation="sigmoid")
])

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

triple_hidden_layer_model.fit(reviews_train_encoded, sentiments_train_array, epochs=6, batch_size=512)
triple_hidden_layer_results = triple_hidden_layer_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 31ms/step - accuracy: 0.6328 - loss: 0.6305
Epoch 2/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8862 - loss: 0.3873
Epoch 3/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9180 - loss: 0.2507
Epoch 4/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9317 - loss: 0.1954
Epoch 5/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9414 - loss: 0.1705
Epoch 6/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9494 - loss: 0.1483
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8770 - loss: 0.3189
```

```
triple_hidden_layer_results
```

```
[0.3151107430458069, 0.8798800110816956]
```

Making Predictions by Using the Trained Data

```
triple_hidden_layer_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[0.14374498],
       [0.99840504],
       [0.7749249 ],
       ...,
       [0.14845441],
       [0.03717259],
       [0.7442619 ]], dtype=float32)
```

3. Model with 32 Hidden Units

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

# Define the model with 32 hidden units
thirty_two_hidden_units_model = keras.Sequential([
    layers.Dense(32, activation="relu"),  # Hidden units 32
    layers.Dense(32, activation="relu"),  # Hidden units 32
    layers.Dense(1, activation="sigmoid")
])

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

# Train the model
thirty_two_hidden_units_history = thirty_two_hidden_units_model.fit(
    partial_reviews_train,
    partial_sentiments_train,
    epochs=20,
    batch_size=512,
    validation_data=(reviews_val, sentiments_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 85ms/step - accuracy: 0.6918 - loss: 0.6023 - val_accuracy: 0.8579 - val_loss: 0.3814
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.8837 - loss: 0.3299 - val_accuracy: 0.8860 - val_loss: 0.2973
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 40ms/step - accuracy: 0.9247 - loss: 0.2263 - val_accuracy: 0.8669 - val_loss: 0.3261
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.9365 - loss: 0.1870 - val_accuracy: 0.8862 - val_loss: 0.2814
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 65ms/step - accuracy: 0.9483 - loss: 0.1498 - val_accuracy: 0.8700 - val_loss: 0.3319
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9543 - loss: 0.1319 - val_accuracy: 0.8766 - val_loss: 0.3154
```

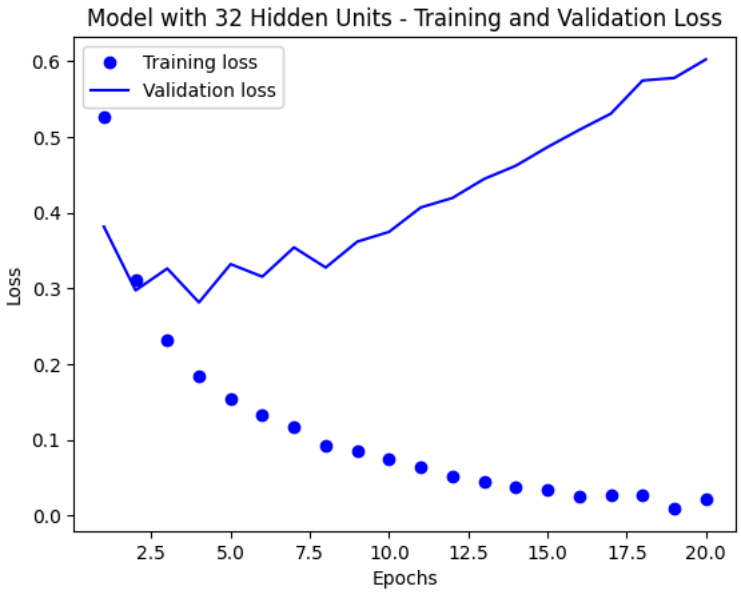
```
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - accuracy: 0.9635 - loss: 0.1112 - val_accuracy: 0.8736 - val_loss: 0.3540
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9721 - loss: 0.0937 - val_accuracy: 0.8852 - val_loss: 0.3275
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9716 - loss: 0.0877 - val_accuracy: 0.8784 - val_loss: 0.3617
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9757 - loss: 0.0744 - val_accuracy: 0.8798 - val_loss: 0.3745
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9853 - loss: 0.0564 - val_accuracy: 0.8717 - val_loss: 0.4067
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9904 - loss: 0.0423 - val_accuracy: 0.8727 - val_loss: 0.4193
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 63ms/step - accuracy: 0.9921 - loss: 0.0372 - val_accuracy: 0.8754 - val_loss: 0.4444
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9942 - loss: 0.0306 - val_accuracy: 0.8728 - val_loss: 0.4618
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9937 - loss: 0.0281 - val_accuracy: 0.8718 - val_loss: 0.4864
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9964 - loss: 0.0214 - val_accuracy: 0.8736 - val_loss: 0.5090
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9959 - loss: 0.0218 - val_accuracy: 0.8716 - val_loss: 0.5305
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9986 - loss: 0.0132 - val_accuracy: 0.8652 - val_loss: 0.5742
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9998 - loss: 0.0099 - val_accuracy: 0.8723 - val_loss: 0.5776
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.9968 - loss: 0.0147 - val_accuracy: 0.8706 - val_loss: 0.6021
```

```
thirty_two_hidden_units_history_dict = thirty_two_hidden_units_history.history
thirty_two_hidden_units_history_dict.keys()
```

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

Plotting the Graph to Show Training and Validation Loss

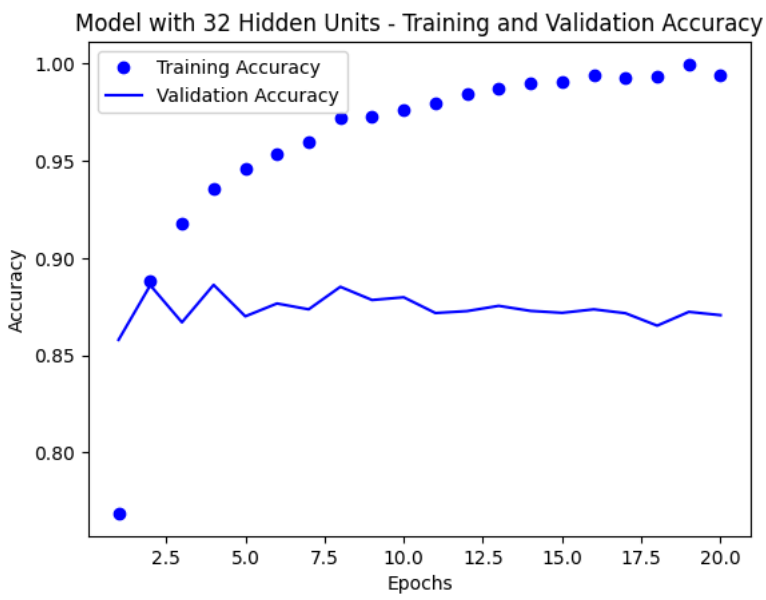
```
import matplotlib.pyplot as plt

thirty_two_hidden_units_history_dict = thirty_two_hidden_units_history.history
training_loss_32_hu = thirty_two_hidden_units_history_dict["loss"]
validation_loss_32_hu = thirty_two_hidden_units_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_32_hu) + 1)

plt.plot(epoch_range, training_loss_32_hu, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_32_hu, "b", label="Validation loss")
plt.title("Model with 32 Hidden Units - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the Accuracy Curve

```
plt.clf()
training_acc_32_hu = thirty_two_hidden_units_history_dict["accuracy"]
validation_acc_32_hu = thirty_two_hidden_units_history_dict["val_accuracy"]
```

```
plt.plot(epoch_range, training_acc_32_hu, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_32_hu, "b", label="Validation Accuracy")
plt.title("Model with 32 Hidden Units - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

```
thirty_two_hidden_units_model = keras.Sequential([
    layers.Dense(32, activation="relu"),  # 32 Hidden Units
    layers.Dense(32, activation="relu"),  # 32 Hidden Units
    layers.Dense(1, activation="sigmoid")
])

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

thirty_two_hidden_units_model.fit(reviews_train_encoded, sentiments_train_array, epochs=3, batch_size=512)
thirty_two_hidden_units_results = thirty_two_hidden_units_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 4s 41ms/step - accuracy: 0.7220 - loss: 0.5545
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 30ms/step - accuracy: 0.9019 - loss: 0.2752
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.9207 - loss: 0.2096
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8858 - loss: 0.2848
```

```
thirty_two_hidden_units_results
```

```
[0.2852748930454254, 0.8870000243186951]
```

Making Predictions by Using the Trained Model

```
thirty_two_hidden_units_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[0.21391854],
       [0.99930394],
       [0.585239  ],
       ...,
       [0.11411382],
       [0.09652016],
       [0.41991216]], dtype=float32)
```

4. Model With 64 Hidden Units

```
sixty_four_hidden_units_history = sixty_four_units_model.fit(
    partial_reviews_train,
    partial_sentiments_train,
```

```
    epochs=20,
    batch_size=512,
    validation_data=(reviews_val, sentiments_val)
)
```

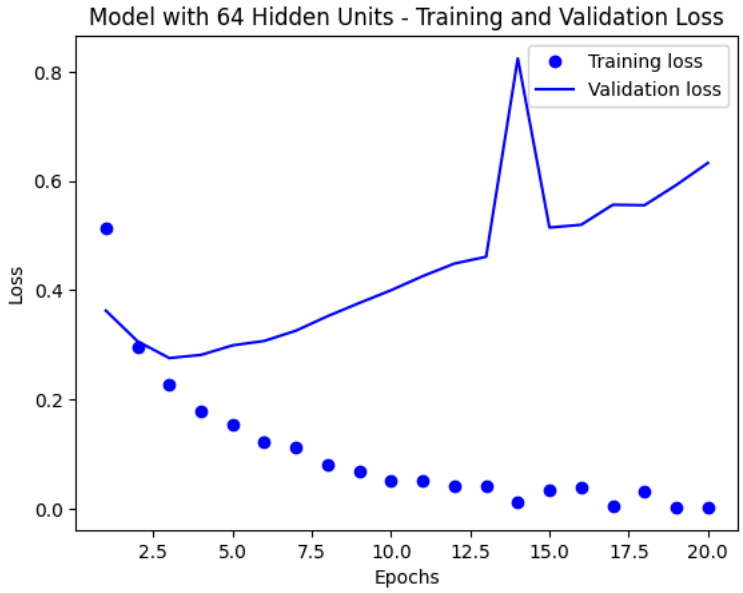
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 92ms/step - accuracy: 0.6756 - loss: 0.5951 - val_accuracy: 0.8572 - val_loss: 0.3624
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 6s 107ms/step - accuracy: 0.8919 - loss: 0.3011 - val_accuracy: 0.8766 - val_loss: 0.3063
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 61ms/step - accuracy: 0.9197 - loss: 0.2243 - val_accuracy: 0.8863 - val_loss: 0.2757
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step - accuracy: 0.9412 - loss: 0.1682 - val_accuracy: 0.8839 - val_loss: 0.2814
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 61ms/step - accuracy: 0.9526 - loss: 0.1376 - val_accuracy: 0.8808 - val_loss: 0.2989
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 61ms/step - accuracy: 0.9608 - loss: 0.1160 - val_accuracy: 0.8822 - val_loss: 0.3070
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 77ms/step - accuracy: 0.9657 - loss: 0.0982 - val_accuracy: 0.8792 - val_loss: 0.3258
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9807 - loss: 0.0702 - val_accuracy: 0.8806 - val_loss: 0.3524
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 62ms/step - accuracy: 0.9839 - loss: 0.0616 - val_accuracy: 0.8772 - val_loss: 0.3765
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - accuracy: 0.9847 - loss: 0.0511 - val_accuracy: 0.8783 - val_loss: 0.3998
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 59ms/step - accuracy: 0.9831 - loss: 0.0571 - val_accuracy: 0.8755 - val_loss: 0.4256
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step - accuracy: 0.9892 - loss: 0.0395 - val_accuracy: 0.8733 - val_loss: 0.4486
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - accuracy: 0.9931 - loss: 0.0274 - val_accuracy: 0.8753 - val_loss: 0.4611
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9993 - loss: 0.0106 - val_accuracy: 0.8302 - val_loss: 0.8242
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 55ms/step - accuracy: 0.9703 - loss: 0.0811 - val_accuracy: 0.8752 - val_loss: 0.5146
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 97ms/step - accuracy: 0.9951 - loss: 0.0187 - val_accuracy: 0.8741 - val_loss: 0.5196
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 61ms/step - accuracy: 0.9998 - loss: 0.0055 - val_accuracy: 0.8738 - val_loss: 0.5563
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 60ms/step - accuracy: 0.9983 - loss: 0.0084 - val_accuracy: 0.8740 - val_loss: 0.5554
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 54ms/step - accuracy: 0.9998 - loss: 0.0034 - val_accuracy: 0.8740 - val_loss: 0.5923
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 60ms/step - accuracy: 0.9999 - loss: 0.0025 - val_accuracy: 0.8726 - val_loss: 0.6328
```

```
sixty_four_hidden_units_history_dict = sixty_four_hidden_units_history.history
sixty_four_hidden_units_history_dict.keys()
```

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

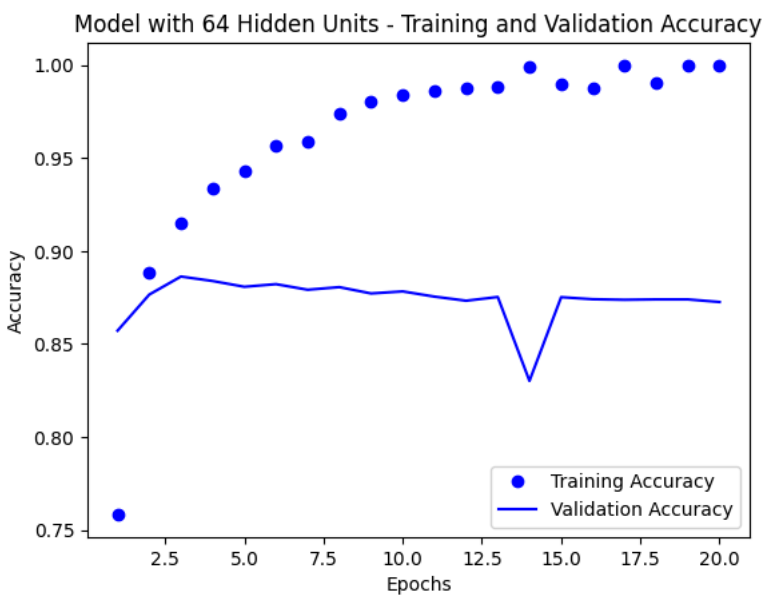
Plotting the Graph to Show Training and Validation Loss

```
import matplotlib.pyplot as plt

sixty_four_hidden_units_history_dict = sixty_four_hidden_units_history.history
training_loss_64_hu = sixty_four_hidden_units_history_dict["loss"]
validation_loss_64_hu = sixty_four_hidden_units_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_64_hu) + 1)

plt.plot(epoch_range, training_loss_64_hu, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_64_hu, "b", label="Validation loss")
plt.title("Model with 64 Hidden Units - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the Accuracy Curve

```
plt.clf()
training_acc_64_hu = sixty_four_hidden_units_history_dict["accuracy"]
validation_acc_64_hu = sixty_four_hidden_units_history_dict["val_accuracy"]

plt.plot(epoch_range, training_acc_64_hu, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_64_hu, "b", label="Validation Accuracy")
plt.title("Model with 64 Hidden Units - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

```
sixty_four_hidden_units_model = keras.Sequential([
    layers.Dense(64, activation="relu"),  # 64 Hidden Units
    layers.Dense(64, activation="relu"),  # 64 Hidden Units
    layers.Dense(1, activation="sigmoid")
])

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

sixty_four_hidden_units_model.fit(reviews_train_encoded, sentiments_train_array, epochs=2, batch_size=512)
sixty_four_hidden_units_results = sixty_four_hidden_units_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - accuracy: 0.7178 - loss: 0.5524
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.8919 - loss: 0.2809
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8851 - loss: 0.2846
```

```
sixty_four_hidden_units_results
```

```
[0.2852901220321655, 0.8853600025177002]
```

Making Predictions by Using the Trained Model

```
sixty_four_hidden_units_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step
array([[0.229705  ],
       [0.9965559 ],
       [0.77154744],
       ...,
       [0.09050532],
       [0.07818089],
       [0.37672368]], dtype=float32)
```

5. Model with Mean Squared Error (MSE) Loss Function

```
mse_loss_history = mse_loss_model.fit(
    partial_reviews_train,
    partial_sentiments_train,
    epochs=20,
    batch_size=512,
    validation_data=(reviews_val, sentiments_val)
)
```

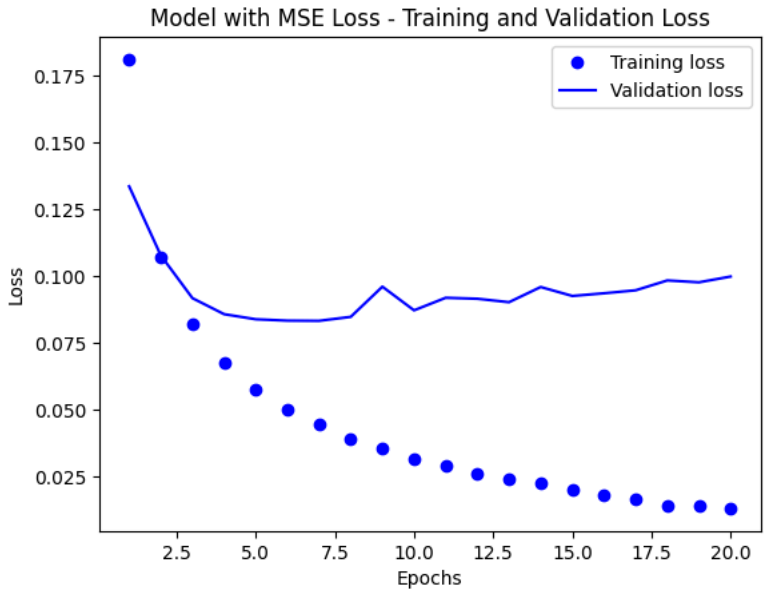
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 5s 129ms/step - accuracy: 0.7072 - loss: 0.2097 - val_accuracy: 0.8582 - val_loss: 0.1336
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 54ms/step - accuracy: 0.8879 - loss: 0.1132 - val_accuracy: 0.8658 - val_loss: 0.1078
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - accuracy: 0.9093 - loss: 0.0843 - val_accuracy: 0.8848 - val_loss: 0.0917
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9270 - loss: 0.0661 - val_accuracy: 0.8877 - val_loss: 0.0857
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9403 - loss: 0.0555 - val_accuracy: 0.8865 - val_loss: 0.0838
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9502 - loss: 0.0487 - val_accuracy: 0.8860 - val_loss: 0.0833
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9559 - loss: 0.0433 - val_accuracy: 0.8856 - val_loss: 0.0832
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 50ms/step - accuracy: 0.9635 - loss: 0.0375 - val_accuracy: 0.8840 - val_loss: 0.0847
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9661 - loss: 0.0343 - val_accuracy: 0.8693 - val_loss: 0.0960
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9677 - loss: 0.0322 - val_accuracy: 0.8820 - val_loss: 0.0872
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9718 - loss: 0.0296 - val_accuracy: 0.8785 - val_loss: 0.0919
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9788 - loss: 0.0242 - val_accuracy: 0.8746 - val_loss: 0.0915
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9802 - loss: 0.0237 - val_accuracy: 0.8775 - val_loss: 0.0902
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9817 - loss: 0.0209 - val_accuracy: 0.8710 - val_loss: 0.0959
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9847 - loss: 0.0193 - val_accuracy: 0.8786 - val_loss: 0.0926
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9877 - loss: 0.0163 - val_accuracy: 0.8774 - val_loss: 0.0936
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - accuracy: 0.9892 - loss: 0.0145 - val_accuracy: 0.8774 - val_loss: 0.0947
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 31ms/step - accuracy: 0.9901 - loss: 0.0140 - val_accuracy: 0.8705 - val_loss: 0.0984
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9897 - loss: 0.0133 - val_accuracy: 0.8730 - val_loss: 0.0977
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9888 - loss: 0.0135 - val_accuracy: 0.8698 - val_loss: 0.0998
```

```
mse_loss_history_dict = mse_loss_history.history
mse_loss_history_dict.keys()
```

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

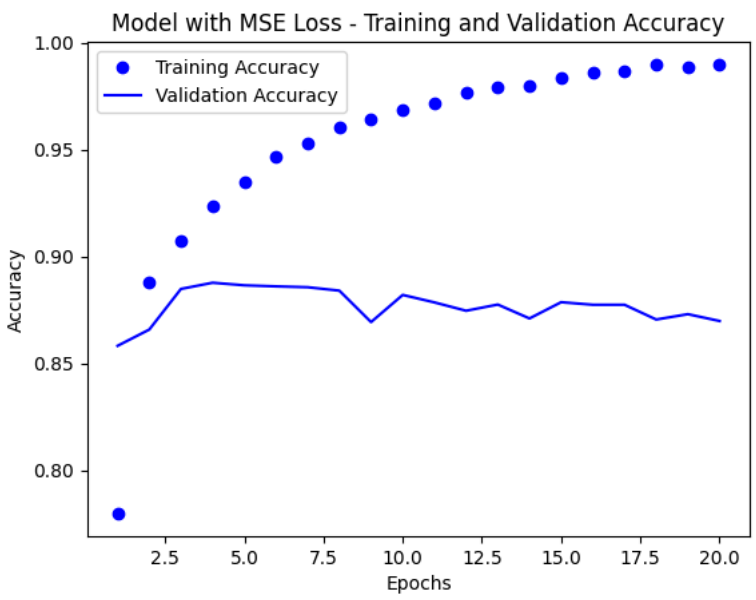
Plotting the Graph to Show Training and Validation Loss

```
import matplotlib.pyplot as plt

mse_loss_history_dict = mse_loss_history.history
training_loss_mse = mse_loss_history_dict["loss"]
validation_loss_mse = mse_loss_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_mse) + 1)

plt.plot(epoch_range, training_loss_mse, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_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 the Accuracy Curve

```
plt.clf()
training_acc_mse = mse_loss_history_dict["accuracy"]
validation_acc_mse = mse_loss_history_dict["val_accuracy"]

plt.plot(epoch_range, training_acc_mse, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_mse, "b", label="Validation Accuracy")
plt.title("Model with MSE Loss - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

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

mse_loss_model.compile(optimizer="rmsprop",
                       loss="mse",  # MSE Loss Function
                       metrics=["accuracy"])

mse_loss_model.fit(reviews_train_encoded, sentiments_train_array, epochs=4, batch_size=512)
mse_loss_results = mse_loss_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.7218 - loss: 0.1951
Epoch 2/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.8988 - loss: 0.0914
Epoch 3/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9211 - loss: 0.0681
Epoch 4/4
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.9331 - loss: 0.0569
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8841 - loss: 0.0863
```

```
mse_loss_results
```

```
[0.08616535365581512, 0.8842800259590149]
```

Making Predictions by Using the Trained Data

```
mse_loss_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[0.16734324],
       [0.999192  ],
       [0.86433715],
       ...,
       [0.12924713],
       [0.08338387],
       [0.45019367]], dtype=float32)
```

6. Model Featuring Tanh Activation

```
tanh_activation_history = model_tanh_activation.fit(
    partial_reviews_train,
    partial_sentiments_train,
    epochs=20,
    batch_size=512,
    validation_data=(reviews_val, sentiments_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 79ms/step - accuracy: 0.7211 - loss: 0.5745 - val_accuracy: 0.8500 - val_loss: 0.3862
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9000 - loss: 0.3071 - val_accuracy: 0.8752 - val_loss: 0.3043
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9247 - loss: 0.2181 - val_accuracy: 0.8892 - val_loss: 0.2743
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9459 - loss: 0.1629 - val_accuracy: 0.8854 - val_loss: 0.2766
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9542 - loss: 0.1359 - val_accuracy: 0.8845 - val_loss: 0.2950
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9706 - loss: 0.0991 - val_accuracy: 0.8826 - val_loss: 0.3179
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9741 - loss: 0.0853 - val_accuracy: 0.8770 - val_loss: 0.3459
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9829 - loss: 0.0670 - val_accuracy: 0.8685 - val_loss: 0.4021
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9857 - loss: 0.0519 - val_accuracy: 0.8737 - val_loss: 0.4125
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - accuracy: 0.9902 - loss: 0.0390 - val_accuracy: 0.8732 - val_loss: 0.4452
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - accuracy: 0.9912 - loss: 0.0376 - val_accuracy: 0.8640 - val_loss: 0.5051
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9912 - loss: 0.0329 - val_accuracy: 0.8704 - val_loss: 0.5091
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9946 - loss: 0.0246 - val_accuracy: 0.8698 - val_loss: 0.5368
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9983 - loss: 0.0141 - val_accuracy: 0.8404 - val_loss: 0.7237
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9807 - loss: 0.0473 - val_accuracy: 0.8655 - val_loss: 0.6029
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9948 - loss: 0.0191 - val_accuracy: 0.8662 - val_loss: 0.6169
```

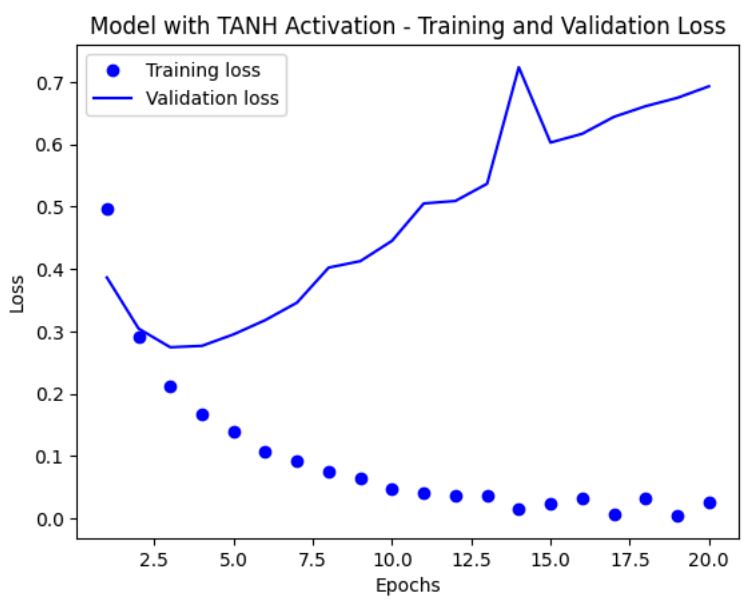
```
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 77ms/step - accuracy: 0.9995 - loss: 0.0064 - val_accuracy: 0.8660 - val_loss: 0.6440
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 78ms/step - accuracy: 0.9966 - loss: 0.0150 - val_accuracy: 0.8662 - val_loss: 0.6612
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.9997 - loss: 0.0035 - val_accuracy: 0.8672 - val_loss: 0.6745
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 38ms/step - accuracy: 0.9984 - loss: 0.0079 - val_accuracy: 0.8663 - val_loss: 0.6931
```

```
tanh_activation_history_dict = tanh_activation_history.history
tanh_activation_history_dict.keys()
```

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

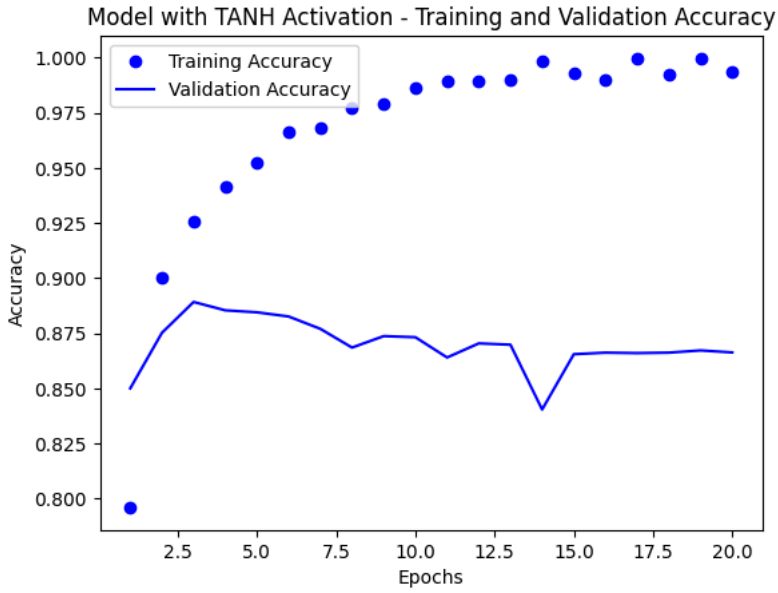
Plotting the Graph to Show Training and Validation Loss

```
import matplotlib.pyplot as plt

tanh_activation_history_dict = tanh_activation_history.history
training_loss_tanh = tanh_activation_history_dict["loss"]
validation_loss_tanh = tanh_activation_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_tanh) + 1)

plt.plot(epoch_range, training_loss_tanh, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_tanh, "b", label="Validation loss")
plt.title("Model with TANH Activation - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the Accuracy Curve

```
plt.clf()
training_acc_tanh = tanh_activation_history_dict["accuracy"]
validation_acc_tanh = tanh_activation_history_dict["val_accuracy"]

plt.plot(epoch_range, training_acc_tanh, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_tanh, "b", label="Validation Accuracy")
plt.title("Model with TANH Activation - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

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

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

tanh_activation_model.fit(reviews_train_encoded, sentiments_train_array, epochs=3, batch_size=512)
tanh_activation_results = tanh_activation_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.7515 - loss: 0.5319
Epoch 2/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.8998 - loss: 0.2703
Epoch 3/3
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.9303 - loss: 0.1869
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8763 - loss: 0.3042
```

```
tanh_activation_results
```

```
[0.29981017112731934, 0.8790000081062317]
```

Making Predictions by Using the Trained Model

```
tanh_activation_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step
array([[0.24050428],
       [0.99665624],
       [0.9081341 ],
       ...,
       [0.15472464],
       [0.10139643],
       [0.7740895 ]], dtype=float32)
```

7. Model With L2 Regularization

```
l2_regularization_history = l2_regularized_model.fit(
    partial_reviews_train,
    partial_sentiments_train,
    epochs=20,
    batch_size=512,
    validation_data=(reviews_val, sentiments_val)
)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 3s 66ms/step - accuracy: 0.6253 - loss: 0.6743 - val_accuracy: 0.8126 - val_loss: 0.5341
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.8675 - loss: 0.4816 - val_accuracy: 0.8533 - val_loss: 0.4370
```

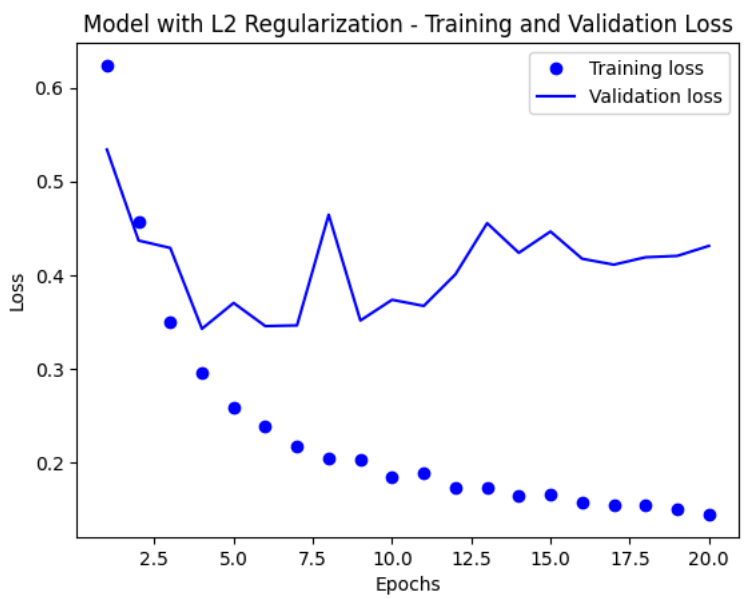
```
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9121 - loss: 0.3632 - val_accuracy: 0.8377 - val_loss: 0.4291
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9241 - loss: 0.3005 - val_accuracy: 0.8876 - val_loss: 0.3429
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9428 - loss: 0.2553 - val_accuracy: 0.8718 - val_loss: 0.3704
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9488 - loss: 0.2353 - val_accuracy: 0.8850 - val_loss: 0.3457
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 48ms/step - accuracy: 0.9561 - loss: 0.2133 - val_accuracy: 0.8826 - val_loss: 0.3465
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 32ms/step - accuracy: 0.9567 - loss: 0.2040 - val_accuracy: 0.8384 - val_loss: 0.4645
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9577 - loss: 0.1981 - val_accuracy: 0.8844 - val_loss: 0.3517
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9664 - loss: 0.1819 - val_accuracy: 0.8785 - val_loss: 0.3738
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9640 - loss: 0.1809 - val_accuracy: 0.8824 - val_loss: 0.3673
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9733 - loss: 0.1674 - val_accuracy: 0.8699 - val_loss: 0.4009
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9712 - loss: 0.1671 - val_accuracy: 0.8562 - val_loss: 0.4555
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9719 - loss: 0.1647 - val_accuracy: 0.8683 - val_loss: 0.4239
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.9733 - loss: 0.1613 - val_accuracy: 0.8617 - val_loss: 0.4466
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.9777 - loss: 0.1528 - val_accuracy: 0.8698 - val_loss: 0.4175
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9826 - loss: 0.1425 - val_accuracy: 0.8751 - val_loss: 0.4113
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9790 - loss: 0.1474 - val_accuracy: 0.8734 - val_loss: 0.4192
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9823 - loss: 0.1411 - val_accuracy: 0.8750 - val_loss: 0.4206
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9817 - loss: 0.1425 - val_accuracy: 0.8716 - val_loss: 0.4313
```

```
l2_regularization_history_dict = l2_regularization_history.history
l2_regularization_history_dict.keys()
```

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

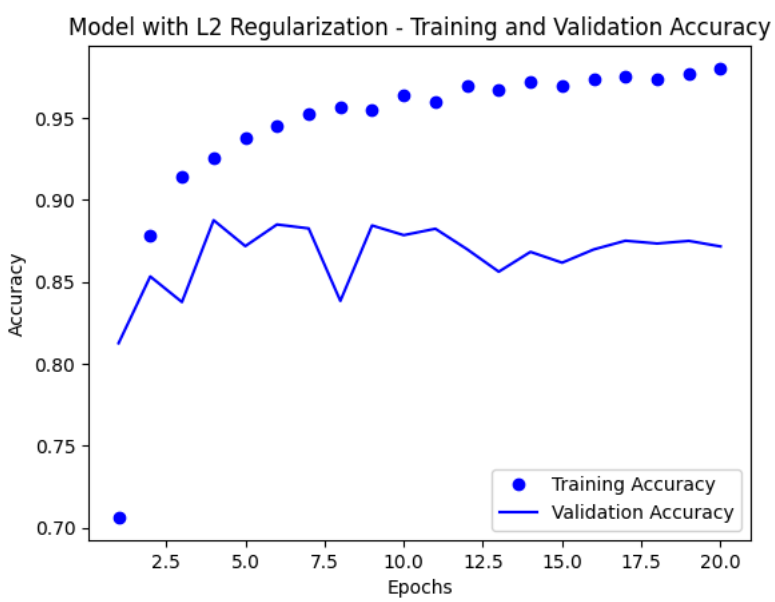
Plotting the Graph to Display Training and Validation Loss

```
import matplotlib.pyplot as plt

l2_regularization_history_dict = l2_regularization_history.history
training_loss_l2 = l2_regularization_history_dict["loss"]
validation_loss_l2 = l2_regularization_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_l2) + 1)

plt.plot(epoch_range, training_loss_l2, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_l2, "b", label="Validation loss")
plt.title("Model with L2 Regularization - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the Accuracy Graph

```
plt.clf()
training_acc_l2 = l2_regularization_history_dict["accuracy"]
validation_acc_l2 = l2_regularization_history_dict["val_accuracy"]

plt.plot(epoch_range, training_acc_l2, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_l2, "b", label="Validation Accuracy")
plt.title("Model with L2 Regularization - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

```
l2_regularized_model = keras.Sequential([
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),  # Applied L2 regularization
    layers.Dense(16, activation="relu", kernel_regularizer=regularizers.l2(0.001)),  # Applied L2 regularization
    layers.Dense(1, activation="sigmoid")
])

l2_regularized_model.compile(optimizer="rmsprop",
                             loss="binary_crossentropy",
                             metrics=["accuracy"])

l2_regularized_model.fit(reviews_train_encoded, sentiments_train_array, epochs=2, batch_size=512)
l2_regularization_results = l2_regularized_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 35ms/step - accuracy: 0.7339 - loss: 0.6045
Epoch 2/2
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.8982 - loss: 0.3431
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8904 - loss: 0.3381
```

```
l2_regularization_results
```

```
[0.338012576103210 45, 0.889959990978241]
```

Making Predictions by Using the Trained Model

```
l2_regularized_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[0.36599338],
       [0.99164385],
       [0.778033  ],
       ...,
       [0.16275862],
       [0.19356214],
       [0.51728755]], dtype=float32)
```

8. Model With Dropout Technique

```
dropout_model_history = model_dropout.fit(
    partial_reviews_train,
    partial_sentiments_train,
    epochs=20,
    batch_size=512,
    validation_data=(reviews_val, sentiments_val)
)
```

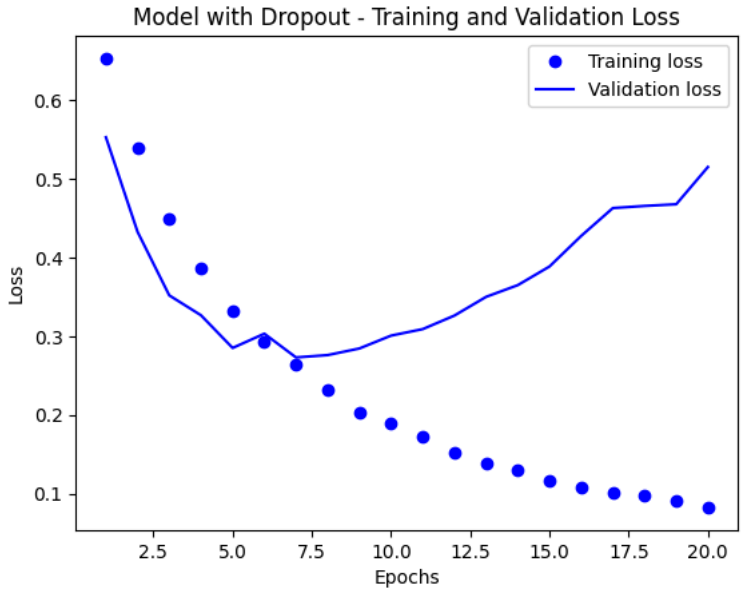
```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - accuracy: 0.5657 - loss: 0.6755 - val_accuracy: 0.8278 - val_loss: 0.5532
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step - accuracy: 0.7291 - loss: 0.5563 - val_accuracy: 0.8671 - val_loss: 0.4328
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.7962 - loss: 0.4663 - val_accuracy: 0.8815 - val_loss: 0.3520
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.8415 - loss: 0.3963 - val_accuracy: 0.8733 - val_loss: 0.3267
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.8650 - loss: 0.3462 - val_accuracy: 0.8886 - val_loss: 0.2851
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.8899 - loss: 0.3003 - val_accuracy: 0.8782 - val_loss: 0.3033
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9061 - loss: 0.2722 - val_accuracy: 0.8903 - val_loss: 0.2731
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - accuracy: 0.9185 - loss: 0.2365 - val_accuracy: 0.8903 - val_loss: 0.2761
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 40ms/step - accuracy: 0.9273 - loss: 0.2066 - val_accuracy: 0.8878 - val_loss: 0.2845
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.9379 - loss: 0.1911 - val_accuracy: 0.8902 - val_loss: 0.3008
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9455 - loss: 0.1699 - val_accuracy: 0.8886 - val_loss: 0.3091
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - accuracy: 0.9519 - loss: 0.1505 - val_accuracy: 0.8899 - val_loss: 0.3263
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9563 - loss: 0.1373 - val_accuracy: 0.8870 - val_loss: 0.3502
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9588 - loss: 0.1238 - val_accuracy: 0.8880 - val_loss: 0.3651
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9618 - loss: 0.1152 - val_accuracy: 0.8825 - val_loss: 0.3888
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.9624 - loss: 0.1108 - val_accuracy: 0.8882 - val_loss: 0.4277
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.9661 - loss: 0.1002 - val_accuracy: 0.8803 - val_loss: 0.4631
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.9664 - loss: 0.0988 - val_accuracy: 0.8860 - val_loss: 0.4658
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 56ms/step - accuracy: 0.9728 - loss: 0.0889 - val_accuracy: 0.8851 - val_loss: 0.4680
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 35ms/step - accuracy: 0.9696 - loss: 0.0827 - val_accuracy: 0.8840 - val_loss: 0.5154
```

```
dropout_model_history_dict = dropout_model_history.history
dropout_model_history_dict.keys()
```

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

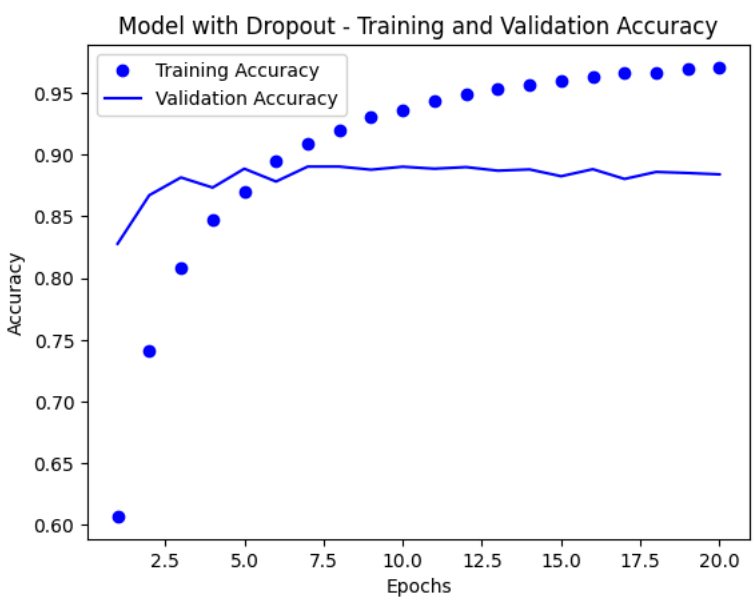
Plotting the Graph to Display Training and Validation Loss

```
import matplotlib.pyplot as plt

dropout_model_history_dict = dropout_model_history.history
training_loss_dropout = dropout_model_history_dict["loss"]
validation_loss_dropout = dropout_model_history_dict["val_loss"]
epoch_range = range(1, len(training_loss_dropout) + 1)

plt.plot(epoch_range, training_loss_dropout, "bo", label="Training loss")
plt.plot(epoch_range, validation_loss_dropout, "b", label="Validation loss")
plt.title("Model with Dropout - Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Plotting the Accuracy Graph

```
plt.clf()
training_acc_dropout = dropout_model_history_dict["accuracy"]
validation_acc_dropout = dropout_model_history_dict["val_accuracy"]

plt.plot(epoch_range, training_acc_dropout, "bo", label="Training Accuracy")
plt.plot(epoch_range, validation_acc_dropout, "b", label="Validation Accuracy")
plt.title("Model with Dropout - Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



Retraining the Model

```
dropout_model = 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")
])

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

```
dropout_model.fit(reviews_train_encoded, sentiments_train_array, epochs=9, batch_size=512)
dropout_model_results = dropout_model.evaluate(reviews_test_encoded, sentiments_test_array)
```

```
Epoch 1/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.6192 - loss: 0.6377
Epoch 2/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 29ms/step - accuracy: 0.8029 - loss: 0.4511
Epoch 3/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.8623 - loss: 0.3608
Epoch 4/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.8888 - loss: 0.3012
Epoch 5/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9061 - loss: 0.2668
Epoch 6/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9207 - loss: 0.2334
Epoch 7/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9292 - loss: 0.2082
Epoch 8/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step - accuracy: 0.9383 - loss: 0.1904
Epoch 9/9
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - accuracy: 0.9424 - loss: 0.1761
782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8790 - loss: 0.3396
```

```
dropout_model_results
```

```
[0.33690690994262695, 0.8804799914360046]
```

Making Predictions by Using the Trained Model

```
dropout_model.predict(reviews_test_encoded)
```

```
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[0.09421223],
       [1.        ],
       [0.9979842 ],
       ...,
       [0.04886092],
       [0.0394343 ],
       [0.8434256 ]], dtype=float32)
```

Models Comparison Analysis

Organizing the Training History of All Models

```
base_model_history_dict = base_model_history.history
base_model_history_dict.keys()

single_hidden_layer_history_dict = single_hidden_layer_history.history
single_hidden_layer_history_dict.keys()

triple_hidden_layer_history_dict = triple_hidden_layer_history.history
triple_hidden_layer_history_dict.keys()

thirty_two_hidden_units_history_dict = thirty_two_hidden_units_history.history
thirty_two_hidden_units_history_dict.keys()

sixty_four_hidden_units_history_dict = sixty_four_hidden_units_history.history
sixty_four_hidden_units_history_dict.keys()

mse_loss_history_dict = mse_loss_history.history
mse_loss_history_dict.keys()

tanh_activation_history_dict = tanh_activation_history.history
tanh_activation_history_dict.keys()

l2_regularization_history_dict = l2_regularization_history.history
l2_regularization_history_dict.keys()

dropout_model_history_dict = dropout_model_history.history
dropout_model_history_dict.keys()
```

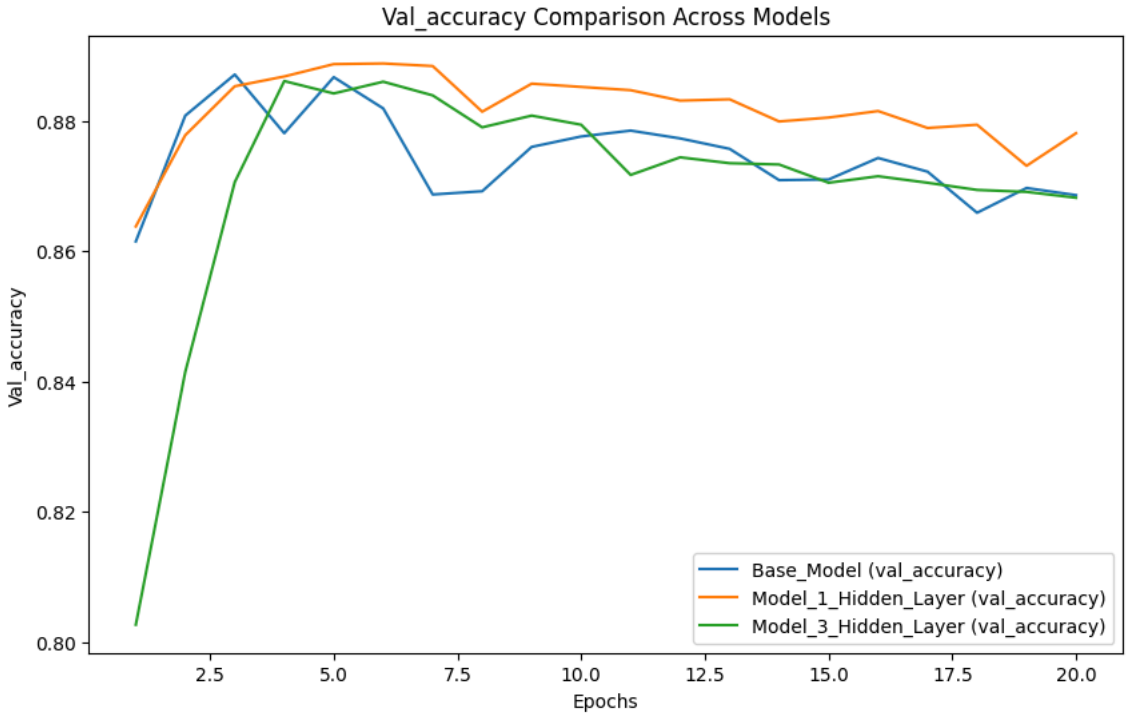
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

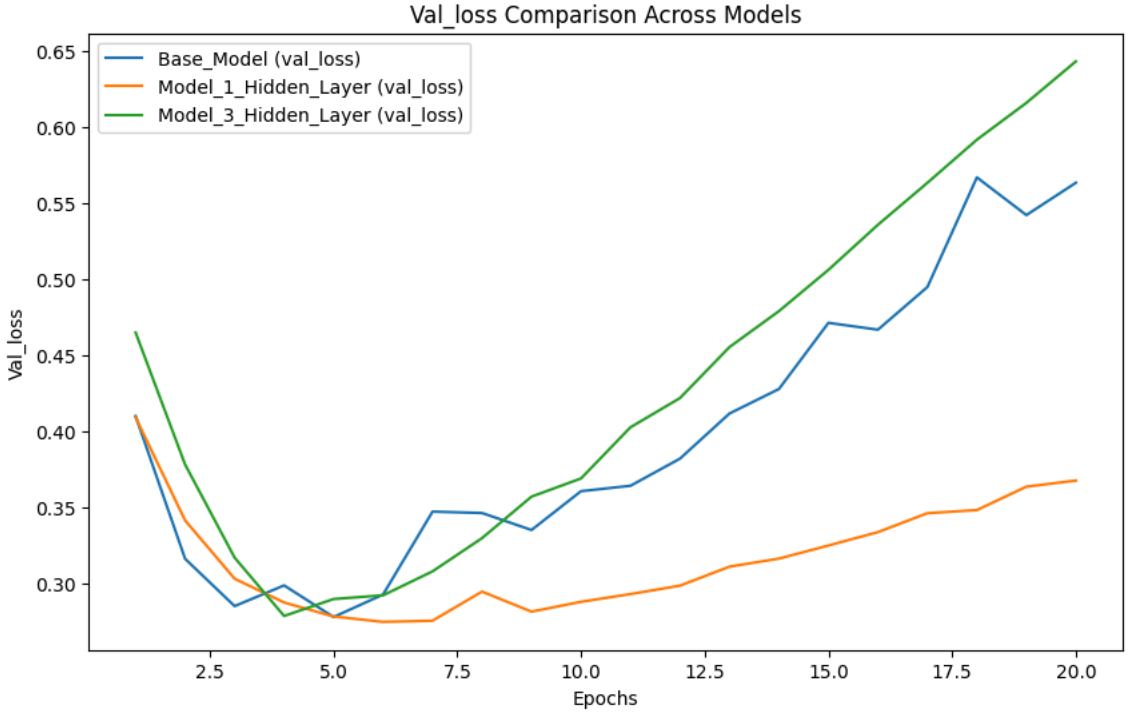
Q1: Comparing Hidden Layers with the Base Model

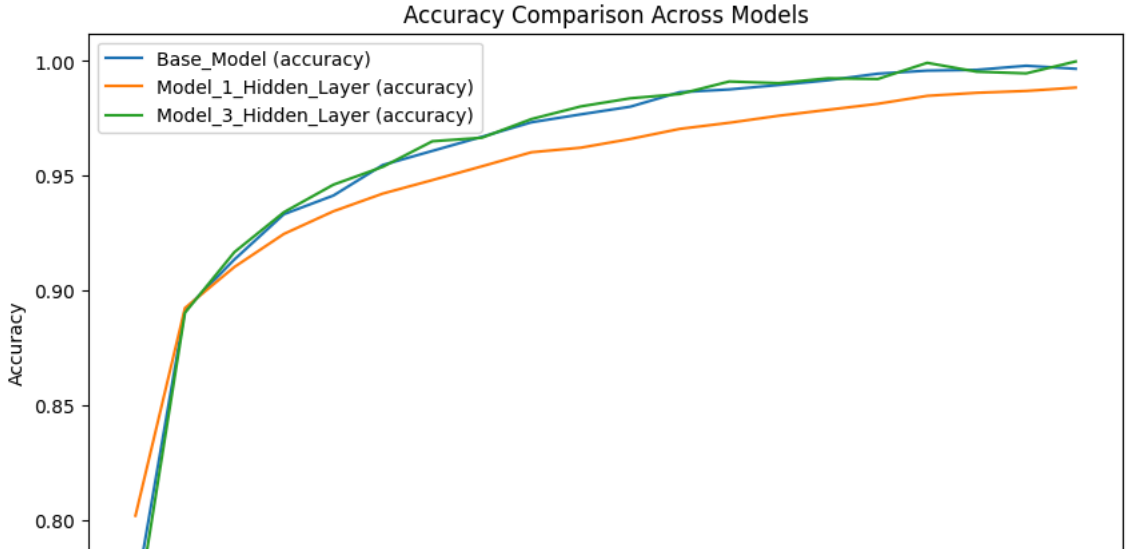
```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
```

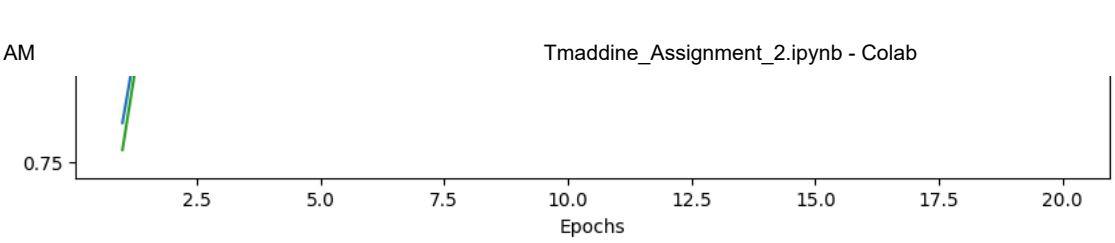
```
        "Base_Model": base_model_history,
        "Model_1_Hidden_Layer": single_hidden_layer_history,
        "Model_3_Hidden_Layer": triple_hidden_layer_history,
    }

    # Extract and display keys of histories
    for model_name, model_history in model_histories.items():
        history_dict = model_history.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_history in model_histories.items():
            metric_values = model_history.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()

    # Plotting validation accuracy
    plot_metrics('val_accuracy')

    # Plotting validation loss
    plot_metrics('val_loss')

    # Plotting training accuracy
    plot_metrics('accuracy')

    # Plotting training loss
    plot_metrics('loss')
```

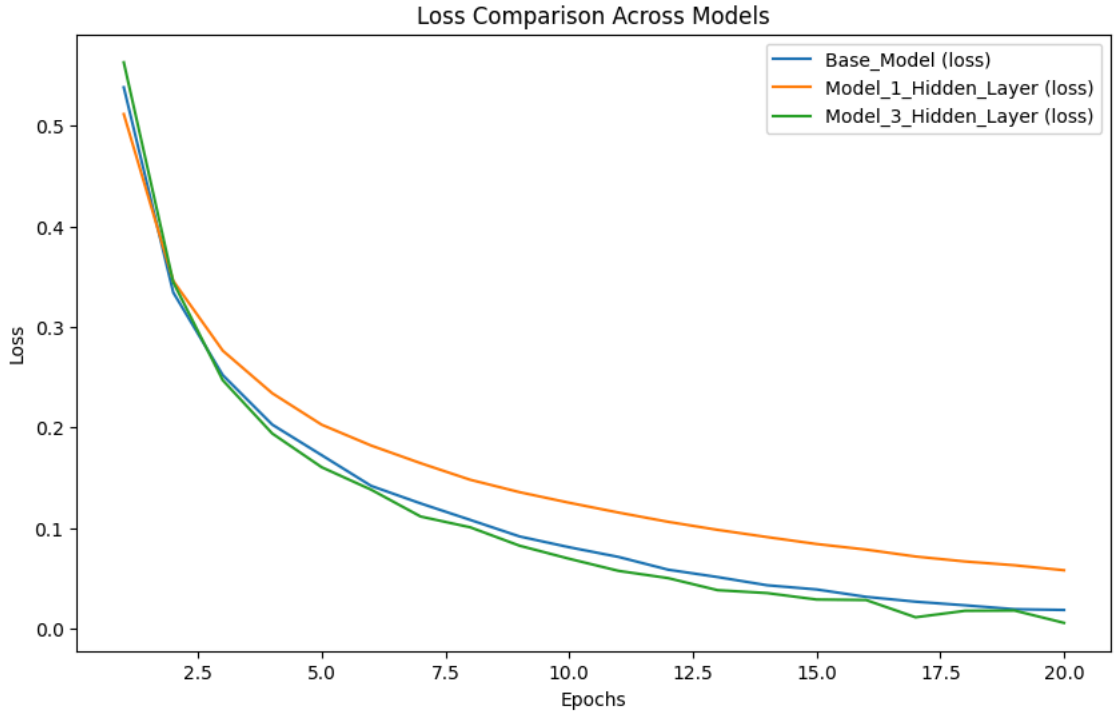
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_1_Hidden_Layer history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_3_Hidden_Layer history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







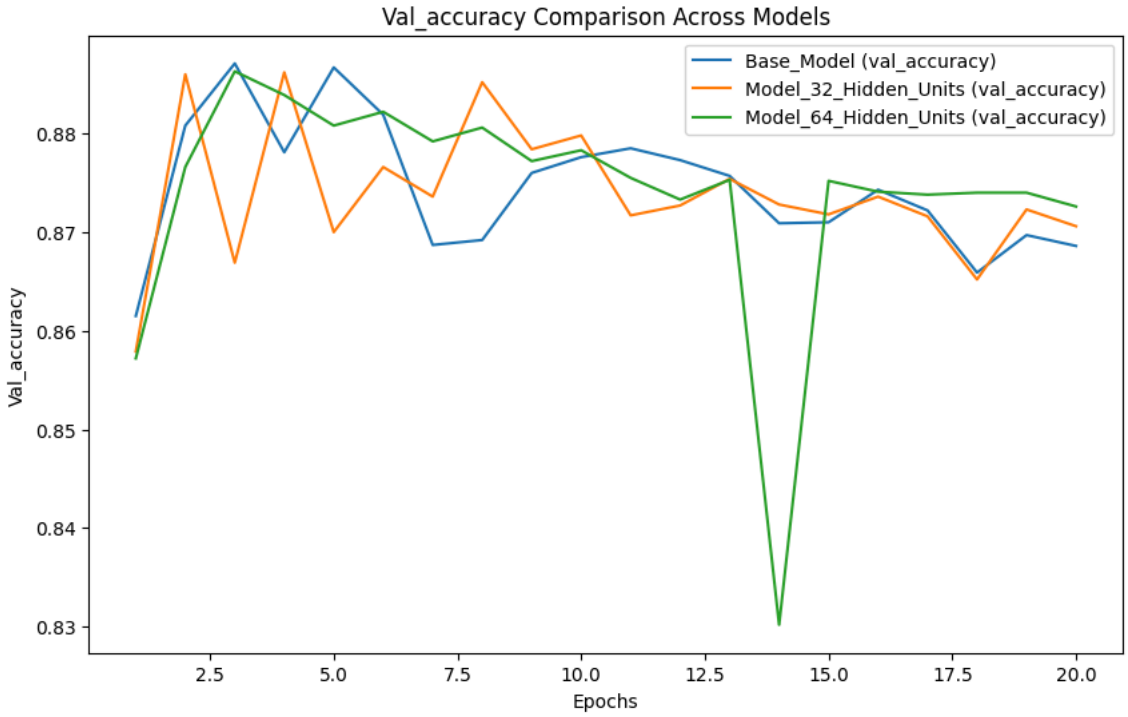


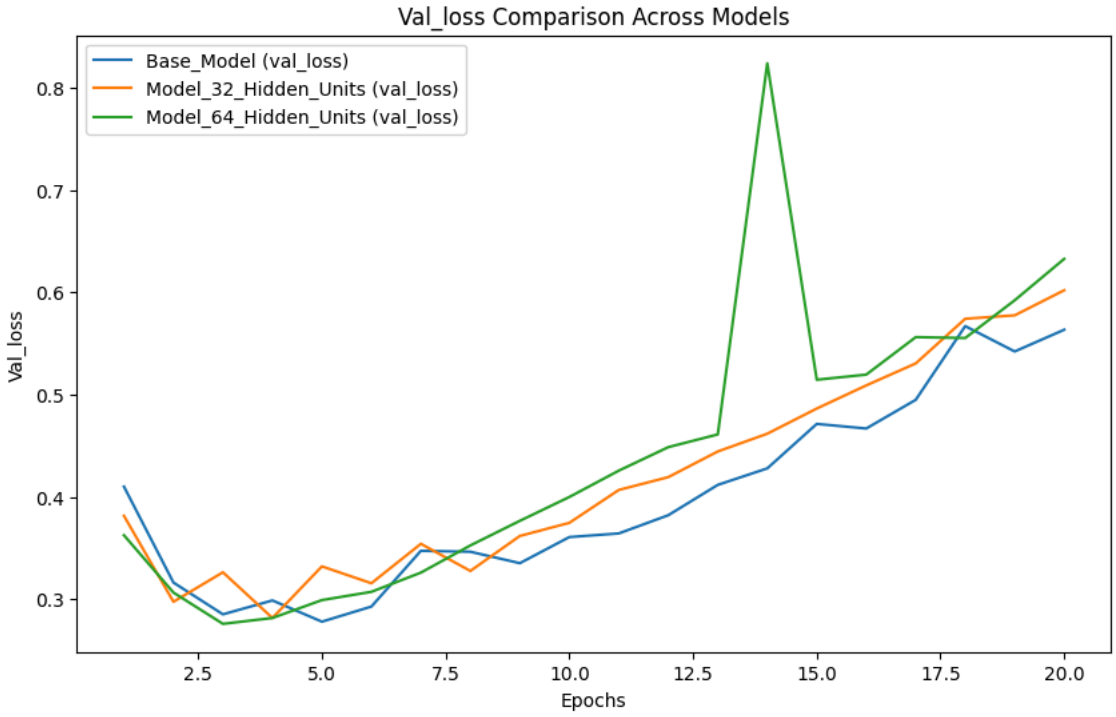


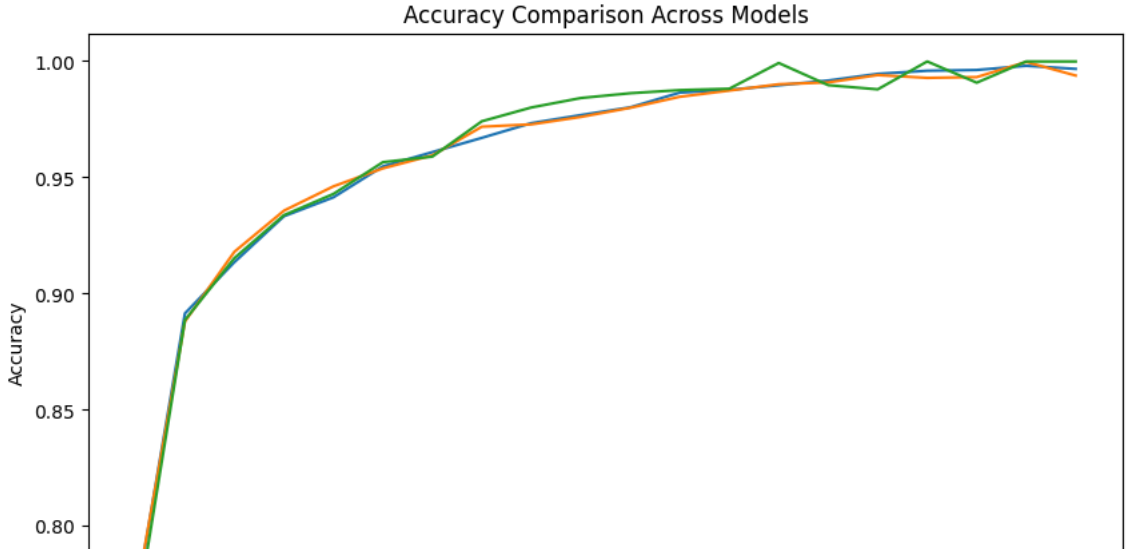
Q2: Comparing the Base Model with Hidden Units of 16, 32, and 64

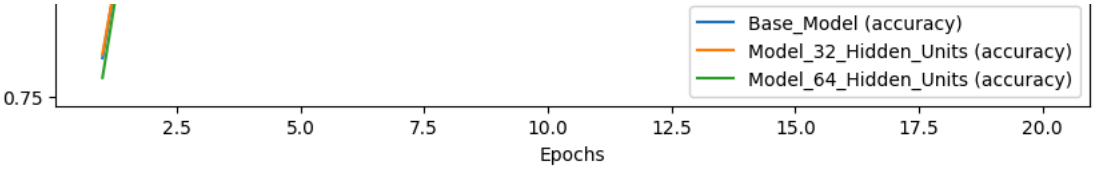
```
import matplotlib.pyplot as plt

# Dictionary of models and their histories
model_histories = {
    "Base_Model": base_model_history,
    "Model_32_Hidden_Units": thirty_two_hidden_units_history,
    "Model_64_Hidden_Units": sixty_four_hidden_units_history,
}

# Extract and display keys of histories
for model_name, model_history in model_histories.items():
    history_dict = model_history.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_history in model_histories.items():
        metric_values = model_history.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()

# Plotting validation accuracy
plot_metrics('val_accuracy')

# Plotting validation loss
plot_metrics('val_loss')

# Plotting training accuracy
plot_metrics('accuracy')

# Plotting training loss
plot_metrics('loss')
```

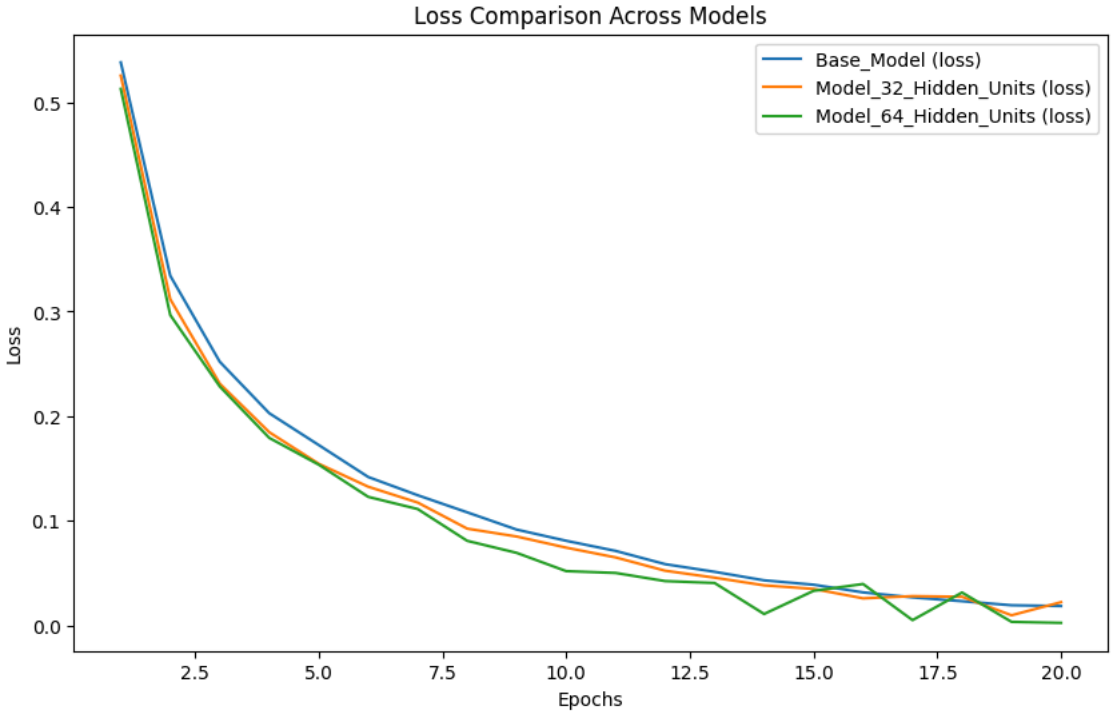
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_32_Hidden_Units history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_64_Hidden_Units history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```







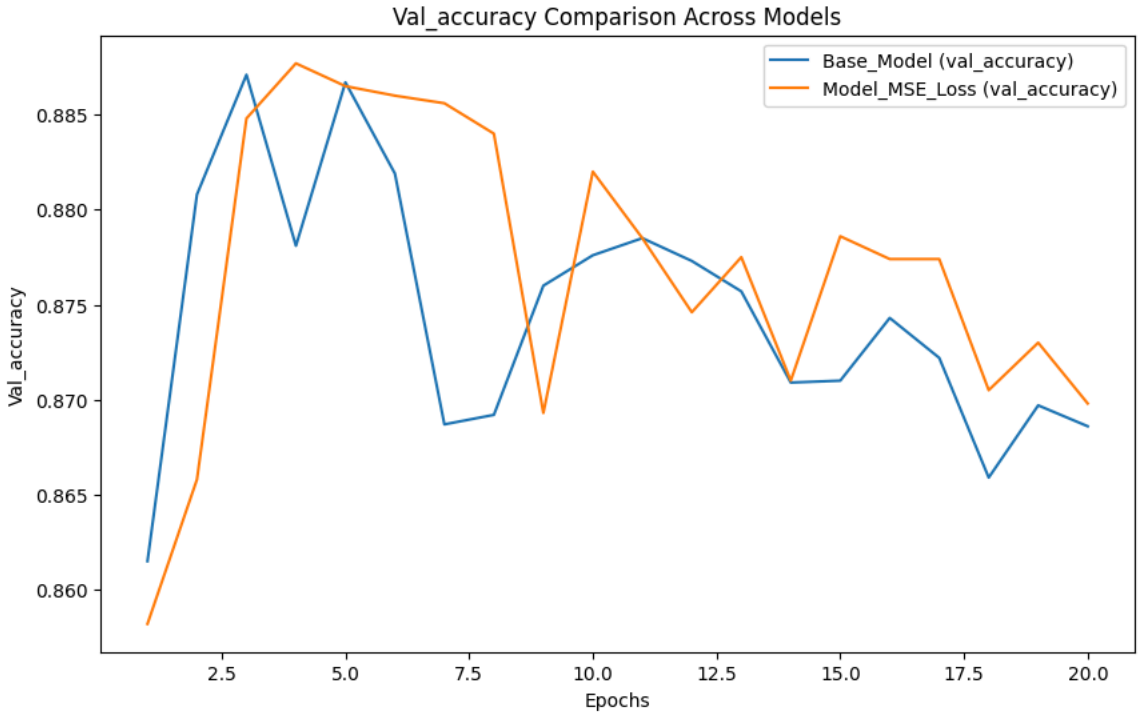


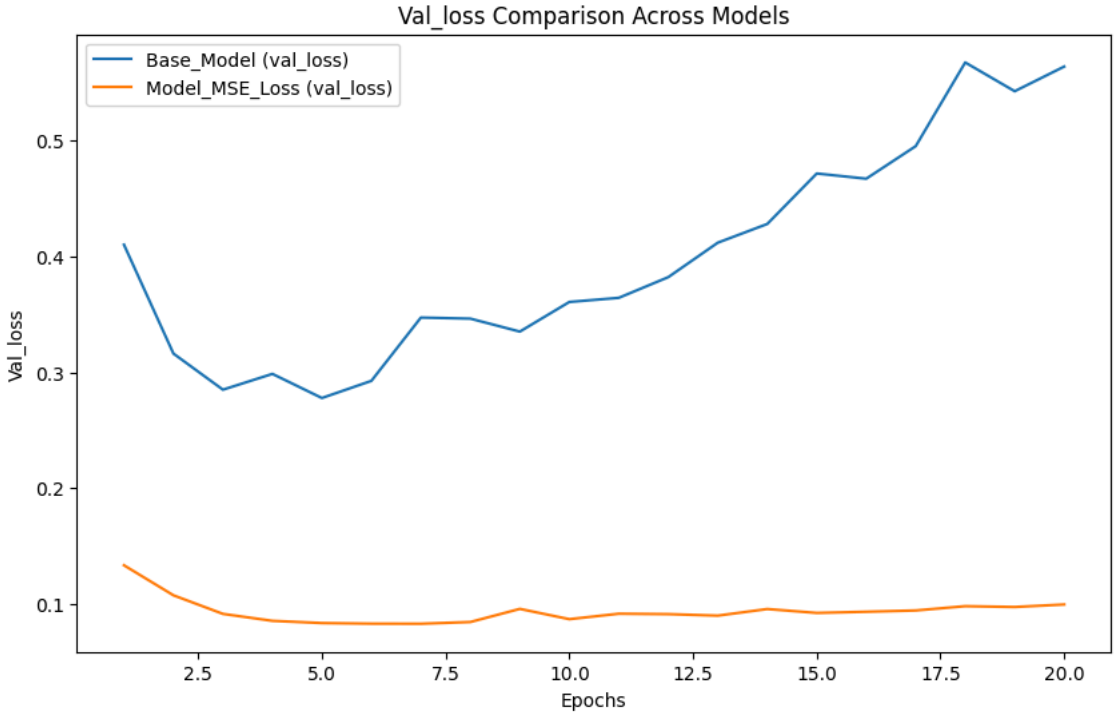


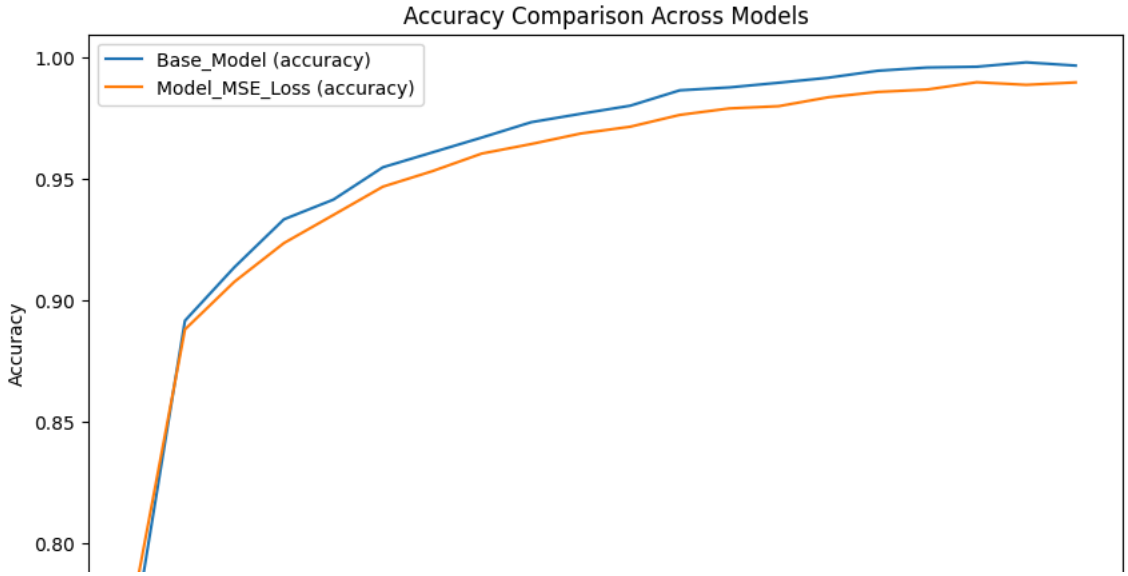
Q3: Comparing the MSE Loss Function

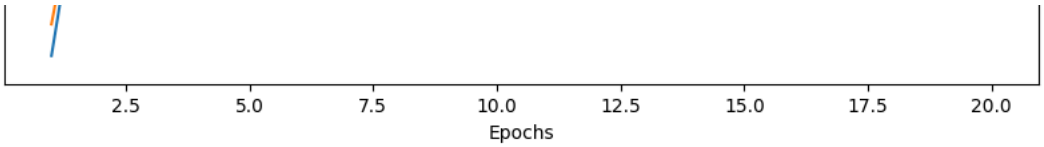
```
import matplotlib.pyplot as plt

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

# Extract and display keys of histories
for model_name, model_history in model_histories.items():
    history_dict = model_history.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_history in model_histories.items():
        metric_values = model_history.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()

# Plotting validation accuracy
plot_metrics('val_accuracy')

# Plotting validation loss
plot_metrics('val_loss')

# Plotting training accuracy
plot_metrics('accuracy')

# Plotting training loss
plot_metrics('loss')
```

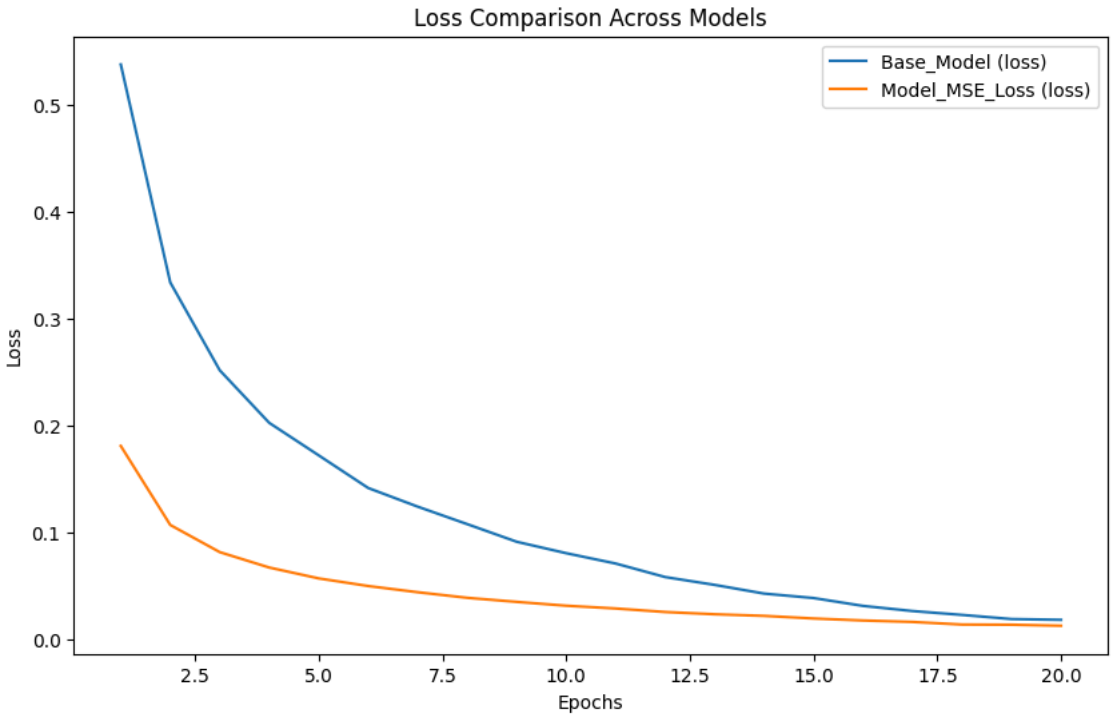
```
Base_Model history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
Model_MSE_Loss history keys: dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```











Q4: Comparing Tanh Activation with the Base Model

```
import matplotlib.pyplot as plt

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

# Extract and display keys of histories
for model_name, model_history in model_histories.items():
    history_dict = model_history.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_history in model_histories.items():
        metric_values = model_history.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()

# Plotting validation accuracy
plot_metrics('val_accuracy')

# Plotting validation loss
plot_metrics('val_loss')

# Plotting training accuracy
plot_metrics('accuracy')

# Plotting training loss
plot_metrics('loss')
```