In [16]:
```
!jupyter nbconvert --to pdf AML_Neural_Network_Assignment_SriAnu(811362334).ipynb
```

```
/bin/bash: -c: line 1: syntax error near unexpected token `('
/bin/bash: -c: line 1: `jupyter nbconvert --to pdf AML_Neural_Network_Assignment_SriAnu(811362334).ipynb'
```

Neural Network Performance Analysis

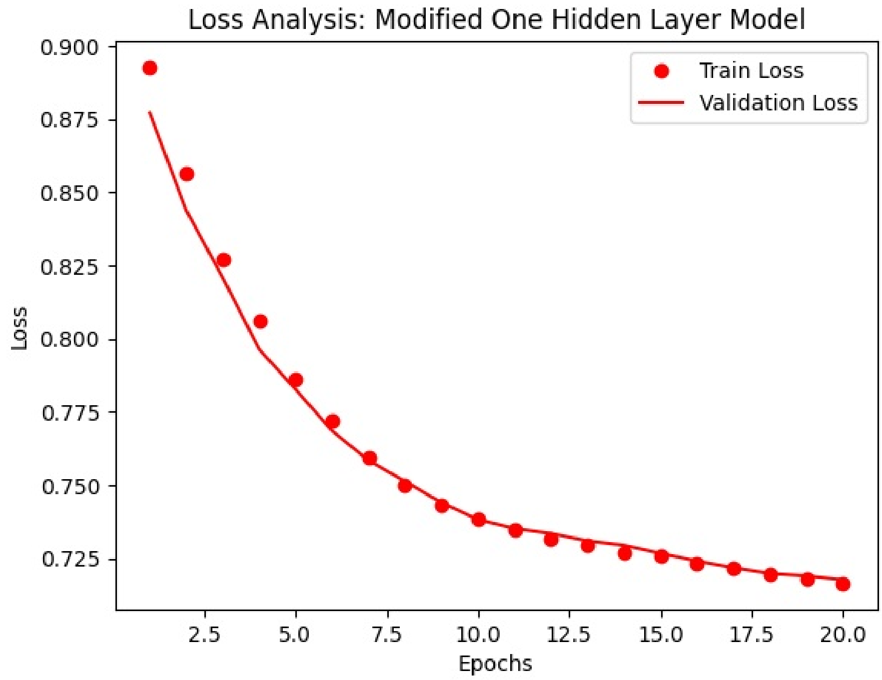
In this notebook, I explore various neural network architectures to predict sentiment in the IMDB dataset. The models were trained with different combinations of hidden layers, hidden units, and loss functions. A detailed analysis is given below. link text

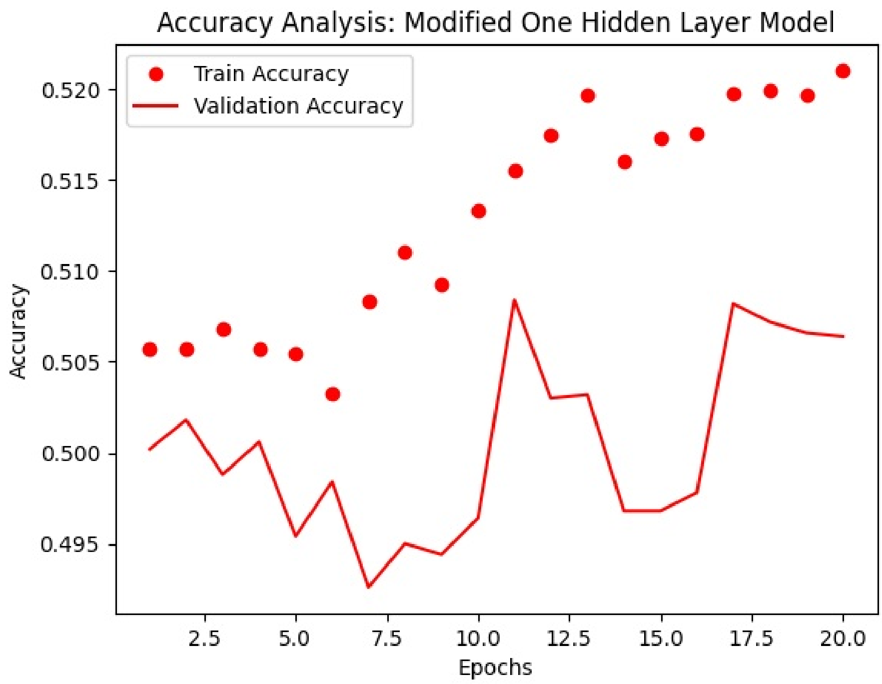
One-Hidden Layer Model

- **Overview**: A single hidden layer, comprised of 32 units with a `tanh` activation function, was trained on the model.
- **Performance**: Training accuracy improved slowly throughout the epochs, but fluctuated across several repeated epochs. Validation accuracy fluctuated around 50%, indicating possible underfitting.
- **Conclusion**: Using just one hidden layer was not very effective, probably due to the simplicity of the model being insufficient for this problem.

In [6]:
```
# Import necessary modules
from tensorflow.keras import models, layers, regularizers
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt

# Load the IMDB dataset
(train_samples, train_labels), (test_samples, test_labels) = imdb.load_data(num_words=10000)

# Preprocess the data (pad sequences to make them the same length)
x_train_mod = pad_sequences(train_samples[:20000], maxlen=256)
y_train_mod = train_labels[:20000]
x_valid = pad_sequences(train_samples[20000:], maxlen=256)
y_valid = train_labels[20000:]

# Define a new model with one hidden layer
simple_nn_model = models.Sequential([
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])

# Compile the model
simple_nn_model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])

# Train the model
history_simple_nn = simple_nn_model.fit(x_train_mod, y_train_mod, epochs=20, batch_size=512, validation_data=(x

# Function to plot training history with new variable names
def plot_new_history(history, graph_title):
    hist_dict = history.history
    loss_data = hist_dict["loss"]
    val_loss_data = hist_dict["val_loss"]
    accuracy_train = hist_dict.get("accuracy", None)
    accuracy_val = hist_dict.get("val_accuracy", None)
    epochs_range = range(1, len(loss_data) + 1)

    # Plot training and validation loss
    plt.plot(epochs_range, loss_data, "ro", label="Train Loss")
    plt.plot(epochs_range, val_loss_data, "r", label="Validation Loss")
    plt.title(f"Loss Analysis: {graph_title}")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

    # If accuracy is available, plot training and validation accuracy
    if accuracy_train and accuracy_val:
        plt.clf()
        plt.plot(epochs_range, accuracy_train, "ro", label="Train Accuracy")
        plt.plot(epochs_range, accuracy_val, "r", label="Validation Accuracy")
        plt.title(f"Accuracy Analysis: {graph_title}")
        plt.xlabel("Epochs")
        plt.ylabel("Accuracy")
        plt.legend()
        plt.show()

# Plot the results for one hidden layer with new names
plot_new_history(history_simple_nn, "Modified One Hidden Layer Model")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.5073 - loss: 0.8992 - val_accuracy: 0.5002 - val_loss: 0.8
771
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5053 - loss: 0.8663 - val_accuracy: 0.5018 - val_loss: 0.84
39
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5029 - loss: 0.8356 - val_accuracy: 0.4988 - val_loss: 0.82
09
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5019 - loss: 0.8117 - val_accuracy: 0.5006 - val_loss: 0.79
64
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5106 - loss: 0.7882 - val_accuracy: 0.4954 - val_loss: 0.78
27
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5075 - loss: 0.7736 - val_accuracy: 0.4984 - val_loss: 0.76
85
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5164 - loss: 0.7604 - val_accuracy: 0.4926 - val_loss: 0.75
84
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5143 - loss: 0.7509 - val_accuracy: 0.4950 - val_loss: 0.75
13
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5046 - loss: 0.7461 - val_accuracy: 0.4944 - val_loss: 0.74
40
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5130 - loss: 0.7390 - val_accuracy: 0.4964 - val_loss: 0.73
83
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5154 - loss: 0.7352 - val_accuracy: 0.5084 - val_loss: 0.73
52
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5199 - loss: 0.7316 - val_accuracy: 0.5030 - val_loss: 0.73
35
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5209 - loss: 0.7298 - val_accuracy: 0.5032 - val_loss: 0.73
10
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5163 - loss: 0.7273 - val_accuracy: 0.4968 - val_loss: 0.72
95
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5148 - loss: 0.7264 - val_accuracy: 0.4968 - val_loss: 0.72
67
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5233 - loss: 0.7234 - val_accuracy: 0.4978 - val_loss: 0.72
41
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5239 - loss: 0.7215 - val_accuracy: 0.5082 - val_loss: 0.72
18
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5220 - loss: 0.7198 - val_accuracy: 0.5072 - val_loss: 0.71
98
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5196 - loss: 0.7185 - val_accuracy: 0.5066 - val_loss: 0.71
90
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5220 - loss: 0.7169 - val_accuracy: 0.5064 - val_loss: 0.71
77
```





Three-Hidden-Layer Model

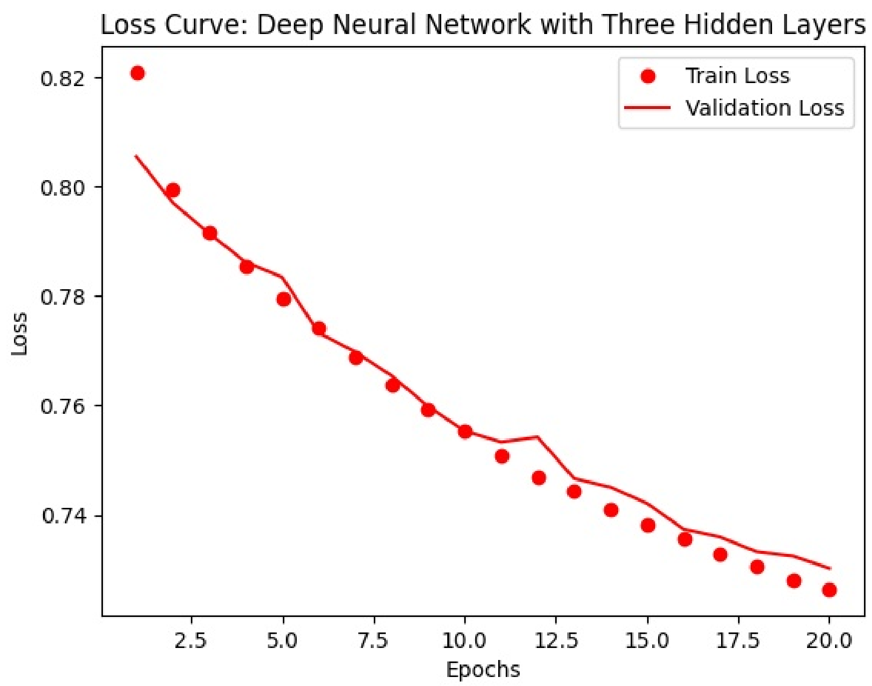
- **Overview**: This model had three hidden layers, each consisting of 32 units.
- **Performance**: Training accuracy was a bit promising over the one-hidden-layer model, although validation accuracy stayed almost at the 50% mark. Training loss decreased, although it was not that much of an improvement compared to validation loss.
- **Conclusion**: Generalization improved with the increase in complexity, but the reasons were most probably overfitting or regularization that was not enough.

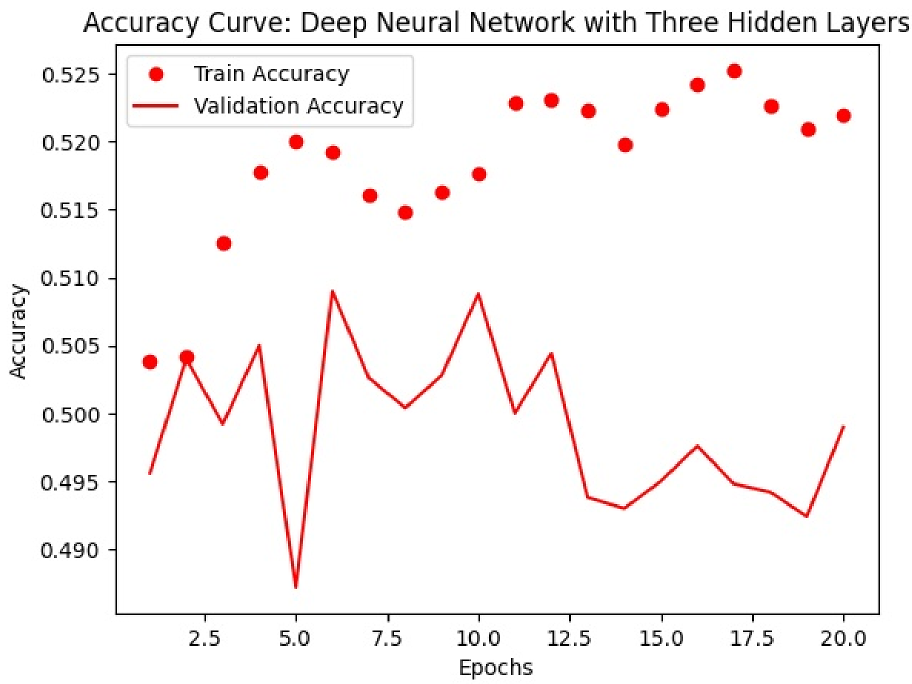
In [9]:

```
# Import necessary modules
from tensorflow.keras import models, layers, regularizers
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt

# Load the IMDB dataset
(train_texts, train_targets), (test_texts, test_targets) = imdb.load_data(num_words=10000)

# Preprocess the data (pad sequences to make them the same length)
x_train_new = pad_sequences(train_texts[:20000], maxlen=256)
y_train_new = train_targets[:20000]
x_validation = pad_sequences(train_texts[20000:], maxlen=256)
y_validation = train_targets[20000:]

# Define a new model with three hidden layers
deep_nn_model = models.Sequential([
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
```

```python
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])

# Compile the model
deep_nn_model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])

# Train the model
history_deep_nn = deep_nn_model.fit(x_train_new, y_train_new, epochs=20, batch_size=512, validation_data=(x_val:

# Function to plot training history with new variable names
def visualize_training(history, title_text):
    hist_info = history.history
    loss_train = hist_info["loss"]
    loss_val = hist_info["val_loss"]
    acc_train = hist_info.get("accuracy", None)
    acc_val = hist_info.get("val_accuracy", None)
    epoch_numbers = range(1, len(loss_train) + 1)

    # Plot training and validation loss
    plt.plot(epoch_numbers, loss_train, "ro", label="Train Loss")  # Changed to green
    plt.plot(epoch_numbers, loss_val, "r", label="Validation Loss")
    plt.title(f"Loss Curve: {title_text}")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

    # If accuracy is available, plot training and validation accuracy
    if acc_train and acc_val:
        plt.clf()
        plt.plot(epoch_numbers, acc_train, "ro", label="Train Accuracy")
        plt.plot(epoch_numbers, acc_val, "r", label="Validation Accuracy")
        plt.title(f"Accuracy Curve: {title_text}")
        plt.xlabel("Epochs")
        plt.ylabel("Accuracy")
        plt.legend()
        plt.show()

# Plot the results for three hidden layers with new names
visualize_training(history_deep_nn, "Deep Neural Network with Three Hidden Layers")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - accuracy: 0.5032 - loss: 0.8373 - val_accuracy: 0.4956 - val_loss: 0.8
055
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5068 - loss: 0.8015 - val_accuracy: 0.5040 - val_loss: 0.79
71
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5165 - loss: 0.7929 - val_accuracy: 0.4992 - val_loss: 0.79
14
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5157 - loss: 0.7871 - val_accuracy: 0.5050 - val_loss: 0.7
862
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5146 - loss: 0.7813 - val_accuracy: 0.4872 - val_loss: 0.7
835
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5217 - loss: 0.7755 - val_accuracy: 0.5090 - val_loss: 0.7
733
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5218 - loss: 0.7695 - val_accuracy: 0.5026 - val_loss: 0.7
698
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5112 - loss: 0.7649 - val_accuracy: 0.5004 - val_loss: 0.76
54
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5210 - loss: 0.7597 - val_accuracy: 0.5028 - val_loss: 0.75
98
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5156 - loss: 0.7562 - val_accuracy: 0.5088 - val_loss: 0.75
53
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5232 - loss: 0.7513 - val_accuracy: 0.5000 - val_loss: 0.75
33
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5193 - loss: 0.7480 - val_accuracy: 0.5044 - val_loss: 0.75
42
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5222 - loss: 0.7452 - val_accuracy: 0.4938 - val_loss: 0.74
66
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5187 - loss: 0.7419 - val_accuracy: 0.4930 - val_loss: 0.74
50
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5265 - loss: 0.7385 - val_accuracy: 0.4950 - val_loss: 0.74
21
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.5265 - loss: 0.7359 - val_accuracy: 0.4976 - val_loss: 0.73
74
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5277 - loss: 0.7331 - val_accuracy: 0.4948 - val_loss: 0.73
59
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5289 - loss: 0.7308 - val_accuracy: 0.4942 - val_loss: 0.73
32
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5233 - loss: 0.7280 - val_accuracy: 0.4924 - val_loss: 0.73
24
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5273 - loss: 0.7263 - val_accuracy: 0.4990 - val_loss: 0.73
02
```





In [ ]:

In [ ]:

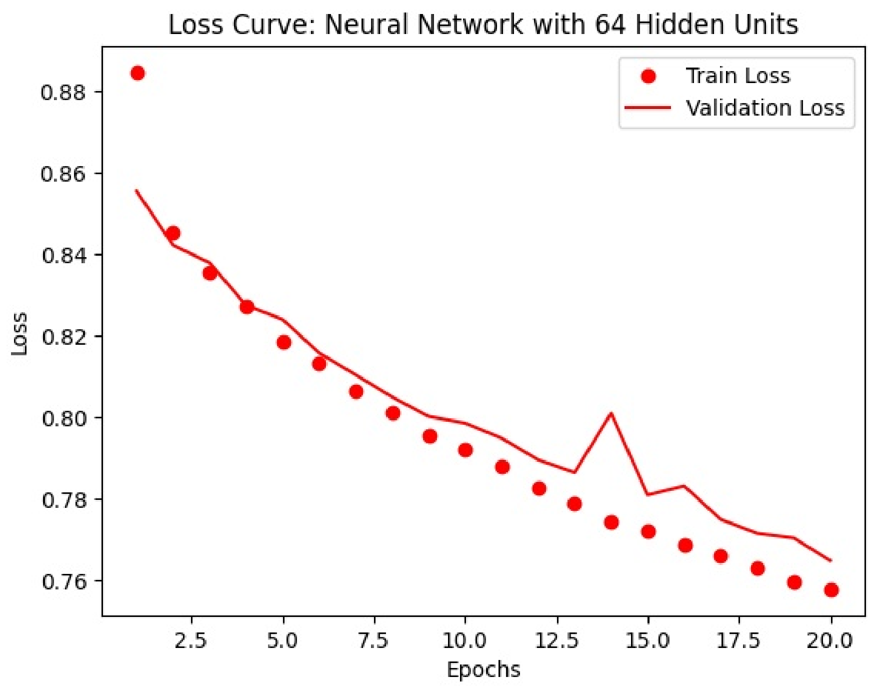
Using More Hidden units (64)

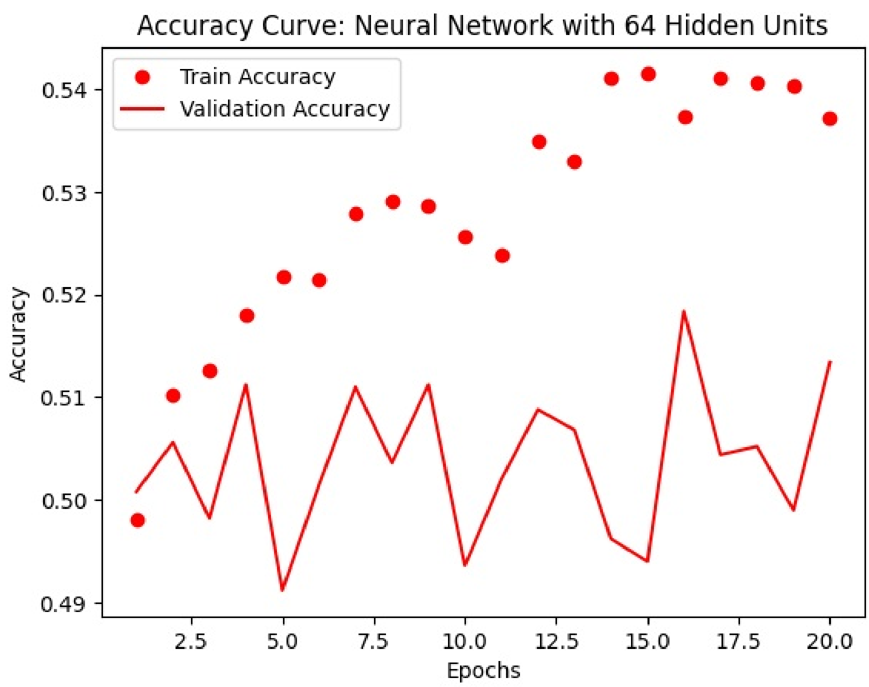
- **Overview**: The model uses 64 units per layer, therefore increasing the capacity of the model.
- **Performance**: [The model] exhibited fluctuations in validation accuracy although training accuracy improved. This model seems to marginally generalize better than its predecessors.
- **Conclusion**: Although increasing the number of hidden units per layer improved performance marginally, more epochs or regularization would assure better generalizability.

In [10]:

```
# Model with 64 hidden units
nn_64_units = models.Sequential([
    layers.Dense(64, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(64, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
nn_64_units.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
history_nn_64 = nn_64_units.fit(x_train_new, y_train_new, epochs=20, batch_size=512, validation_data=(x_validat

# Plot the results for 64 hidden units
visualize_training(history_nn_64, "Neural Network with 64 Hidden Units")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.4953 - loss: 0.9080 - val_accuracy: 0.5008 - val_loss: 0.8
554
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5104 - loss: 0.8473 - val_accuracy: 0.5056 - val_loss: 0.84
22
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5169 - loss: 0.8373 - val_accuracy: 0.4982 - val_loss: 0.83
79
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5187 - loss: 0.8281 - val_accuracy: 0.5112 - val_loss: 0.82
74
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5244 - loss: 0.8198 - val_accuracy: 0.4912 - val_loss: 0.82
40
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5260 - loss: 0.8132 - val_accuracy: 0.5014 - val_loss: 0.81
57
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.5244 - loss: 0.8086 - val_accuracy: 0.5110 - val_loss: 0.81
04
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.5340 - loss: 0.8013 - val_accuracy: 0.5036 - val_loss: 0.80
49
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5308 - loss: 0.7964 - val_accuracy: 0.5112 - val_loss: 0.80
02
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5293 - loss: 0.7919 - val_accuracy: 0.4936 - val_loss: 0.79
85
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.5266 - loss: 0.7882 - val_accuracy: 0.5020 - val_loss: 0.79
49
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.5361 - loss: 0.7838 - val_accuracy: 0.5088 - val_loss: 0.7
895
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.5387 - loss: 0.7791 - val_accuracy: 0.5068 - val_loss: 0.7
864
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.5437 - loss: 0.7738 - val_accuracy: 0.4962 - val_loss: 0.8
010
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5430 - loss: 0.7723 - val_accuracy: 0.4940 - val_loss: 0.7
809
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.5273 - loss: 0.7703 - val_accuracy: 0.5184 - val_loss: 0.78
31
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5410 - loss: 0.7684 - val_accuracy: 0.5044 - val_loss: 0.77
49
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5431 - loss: 0.7621 - val_accuracy: 0.5052 - val_loss: 0.77
15
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5461 - loss: 0.7594 - val_accuracy: 0.4990 - val_loss: 0.77
04
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5392 - loss: 0.7580 - val_accuracy: 0.5134 - val_loss: 0.76
48
```



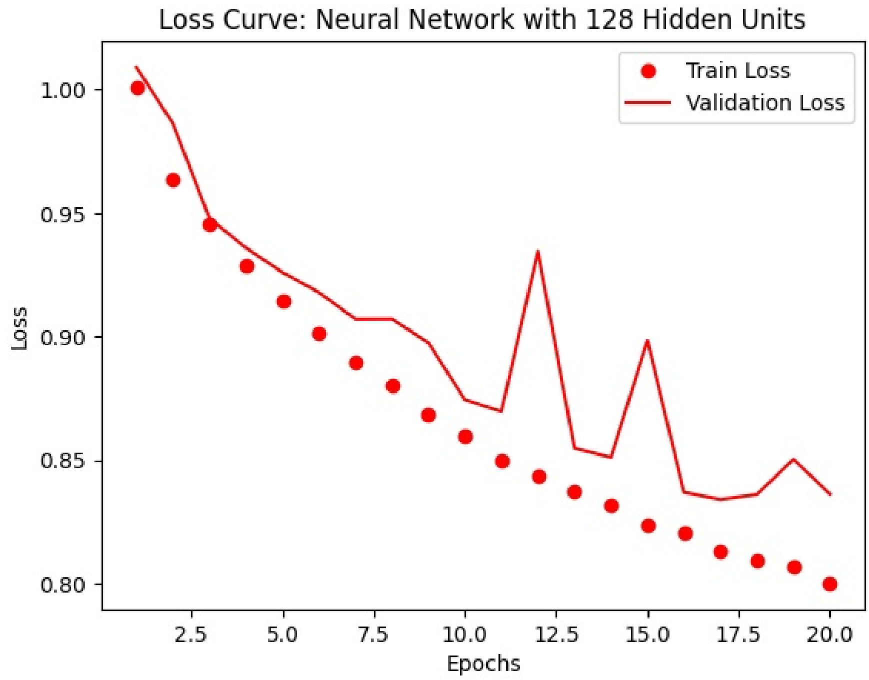


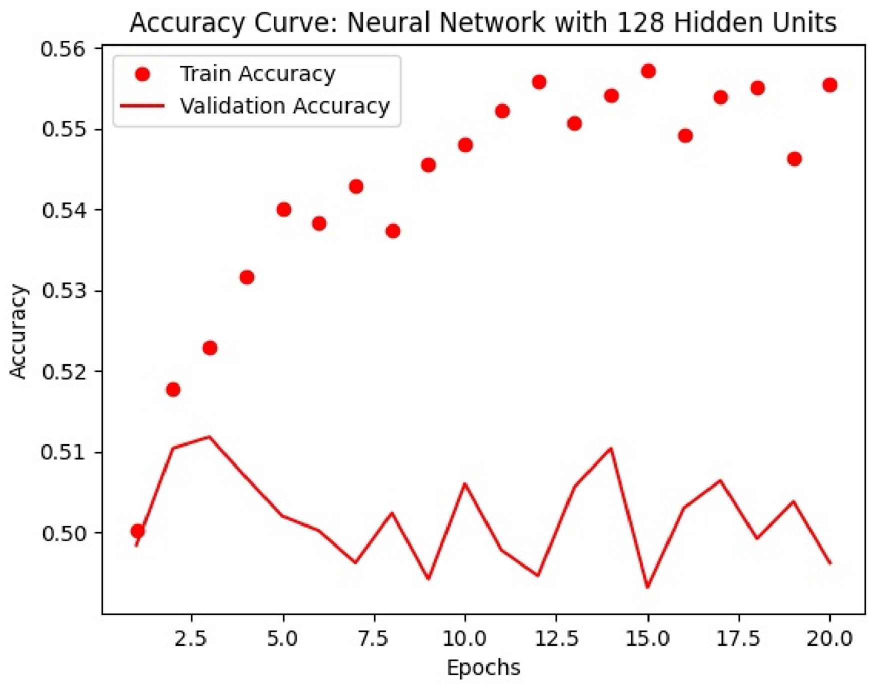
Model with 128 hidden units

- Overview: A model structure using 128 units per layer is proposed, which would allow its capacity to be increased.
- Performance: The model showed small fluctuations in validation accuracy although training accuracy improved. The model appears to generalize moderately better compared to its predecessors.
- Conclusion: Even though increased hidden units per layer improved performance marginally, more epochs or some sort of regularization would assist with generalization.

In [11]:
```
# Model with 128 hidden units
nn_128_units = models.Sequential([
    layers.Dense(128, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(128, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
nn_128_units.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
history_nn_128 = nn_128_units.fit(x_train_new, y_train_new, epochs=20, batch_size=512, validation_data=(x_valid

# Plot the results for 128 hidden units
visualize_training(history_nn_128, "Neural Network with 128 Hidden Units")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.4996 - loss: 1.0223 - val_accuracy: 0.4984 - val_loss: 1.0
090
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 18ms/step - accuracy: 0.5192 - loss: 0.9687 - val_accuracy: 0.5104 - val_loss: 0.9
862
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step - accuracy: 0.5152 - loss: 0.9522 - val_accuracy: 0.5118 - val_loss: 0.9
479
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.5320 - loss: 0.9311 - val_accuracy: 0.5068 - val_loss: 0.9
358
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5398 - loss: 0.9184 - val_accuracy: 0.5020 - val_loss: 0.92
59
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5456 - loss: 0.9010 - val_accuracy: 0.5002 - val_loss: 0.9
178
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5488 - loss: 0.8904 - val_accuracy: 0.4962 - val_loss: 0.9
070
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5341 - loss: 0.8834 - val_accuracy: 0.5024 - val_loss: 0.9
071
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5486 - loss: 0.8698 - val_accuracy: 0.4942 - val_loss: 0.89
74
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.5480 - loss: 0.8606 - val_accuracy: 0.5060 - val_loss: 0.8
743
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5569 - loss: 0.8495 - val_accuracy: 0.4978 - val_loss: 0.86
96
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5573 - loss: 0.8447 - val_accuracy: 0.4946 - val_loss: 0.9
345
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5540 - loss: 0.8418 - val_accuracy: 0.5056 - val_loss: 0.85
47
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5609 - loss: 0.8311 - val_accuracy: 0.5104 - val_loss: 0.8
510
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.5616 - loss: 0.8237 - val_accuracy: 0.4932 - val_loss: 0.89
83
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.5491 - loss: 0.8242 - val_accuracy: 0.5030 - val_loss: 0.8
368
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5579 - loss: 0.8122 - val_accuracy: 0.5064 - val_loss: 0.83
40
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.5580 - loss: 0.8085 - val_accuracy: 0.4992 - val_loss: 0.8
360
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5465 - loss: 0.8069 - val_accuracy: 0.5038 - val_loss: 0.85
02
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.5549 - loss: 0.8018 - val_accuracy: 0.4962 - val_loss: 0.83
61
```



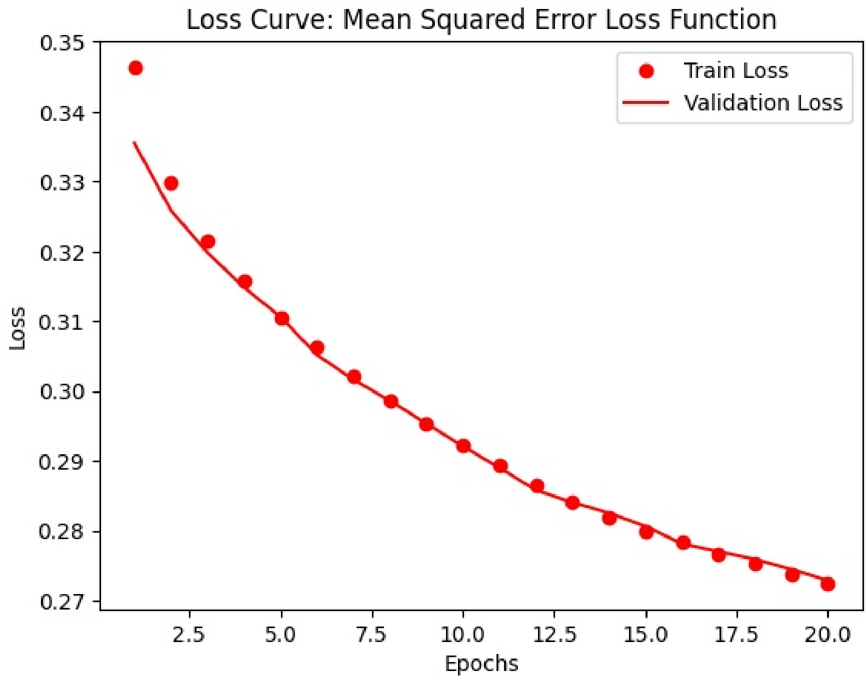


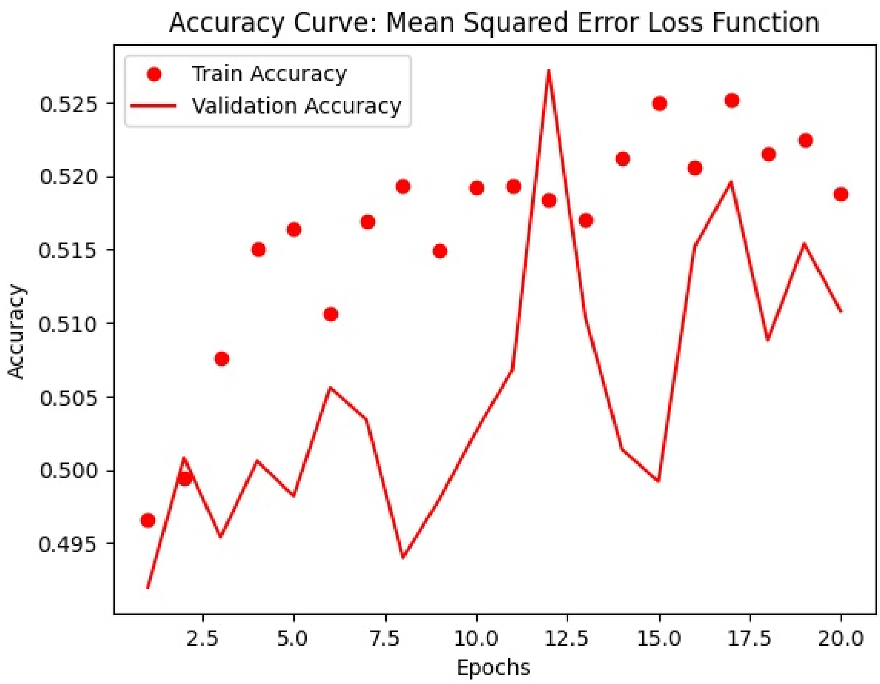
Model with MSE Loss Function

- **Overview**: The default `binary_crossentropy` loss was replaced by an `mse` loss in this model.
- **Performance**: The training loss decreased steadily. However, validation accuracy was oscillatory in its values, with the model performing similarly to the binary-crossentropy model without much gain.
- **Conclusion**: MSE as a loss function does not significantly improve the performance, since binarycrossentropy is the better choice when dealing with binary classifications.

In [12]:
```
# Model with MSE loss function
nn_mse_loss = models.Sequential([
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(32, activation="tanh", kernel_regularizer=regularizers.l2(0.001)),
    layers.Dense(1, activation="sigmoid")
])
nn_mse_loss.compile(optimizer="rmsprop", loss="mse", metrics=["accuracy"])
history_nn_mse = nn_mse_loss.fit(x_train_new, y_train_new, epochs=20, batch_size=512, validation_data=(x_valida

# Plot the results for MSE loss
visualize_training(history_nn_mse, "Mean Squared Error Loss Function")
```

```
Epoch 1/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.4949 - loss: 0.3531 - val_accuracy: 0.4920 - val_loss: 0.3
355
Epoch 2/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.4981 - loss: 0.3328 - val_accuracy: 0.5008 - val_loss: 0.32
59
Epoch 3/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5082 - loss: 0.3230 - val_accuracy: 0.4954 - val_loss: 0.31
98
Epoch 4/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5120 - loss: 0.3168 - val_accuracy: 0.5006 - val_loss: 0.31
48
Epoch 5/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5130 - loss: 0.3117 - val_accuracy: 0.4982 - val_loss: 0.31
06
Epoch 6/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5138 - loss: 0.3068 - val_accuracy: 0.5056 - val_loss: 0.30
52
Epoch 7/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5124 - loss: 0.3032 - val_accuracy: 0.5034 - val_loss: 0.30
16
Epoch 8/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5226 - loss: 0.2991 - val_accuracy: 0.4940 - val_loss: 0.29
86
Epoch 9/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5131 - loss: 0.2959 - val_accuracy: 0.4980 - val_loss: 0.29
53
Epoch 10/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5211 - loss: 0.2930 - val_accuracy: 0.5026 - val_loss: 0.29
21
Epoch 11/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5254 - loss: 0.2896 - val_accuracy: 0.5068 - val_loss: 0.28
90
Epoch 12/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5196 - loss: 0.2870 - val_accuracy: 0.5272 - val_loss: 0.28
58
Epoch 13/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5140 - loss: 0.2847 - val_accuracy: 0.5104 - val_loss: 0.28
41
Epoch 14/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5275 - loss: 0.2819 - val_accuracy: 0.5014 - val_loss: 0.28
25
Epoch 15/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5268 - loss: 0.2801 - val_accuracy: 0.4992 - val_loss: 0.28
07
Epoch 16/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5200 - loss: 0.2789 - val_accuracy: 0.5152 - val_loss: 0.27
81
Epoch 17/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5247 - loss: 0.2771 - val_accuracy: 0.5196 - val_loss: 0.27
70
Epoch 18/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5170 - loss: 0.2757 - val_accuracy: 0.5088 - val_loss: 0.27
59
Epoch 19/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5203 - loss: 0.2743 - val_accuracy: 0.5154 - val_loss: 0.27
45
Epoch 20/20
40/40 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5217 - loss: 0.2727 - val_accuracy: 0.5108 - val_loss: 0.27
29
```





Overall Summary

In this experiment, changing the number of hidden layers, units, and loss function gave different results. Though generally increasing complexity results in better training performance, reducing validation accuracy might indicate overfitting or inadequate regularization.