# Assignment 2 - Neural Networks

This notebook explores different configurations of a neural network for classifying IMDB movie reviews.

## Step 1: Load and Preprocess Data

We will load the IMDB dataset and preprocess it for training.

In [18]:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Load the IMDB dataset
from tensorflow.keras.datasets import imdb

num_words = 10000  # Restrict to top 10,000 words
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=num_words)

# Function to vectorize data (one-hot encoding)
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

# Preprocess data
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

# Split training data into training and validation sets
```

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

print(f"Training data shape: {x_train.shape}")
print(f"Validation data shape: {x_val.shape}")
print(f"Test data shape: {x_test.shape}")
```

```
Training data shape: (25000, 10000)
Validation data shape: (10000, 10000)
Test data shape: (25000, 10000)
```

## Step 2: Define Function to Build Models

In [19]:

```python
# Define the model function
def build_model(hidden_layers=2, hidden_units=16, activation="relu", loss_fn="binary_crossentropy"):
    model = keras.Sequential()
    model.add(layers.Dense(hidden_units, activation=activation, input_shape=(10000,)))
    for _ in range(hidden_layers - 1):
        model.add(layers.Dense(hidden_units, activation=activation))
    model.add(layers.Dense(1, activation="sigmoid"))  # Output layer

    model.compile(optimizer="rmsprop", loss=loss_fn, metrics=["accuracy"])
    return model
```

## Step 3: Experimenting with Different Architectures

We will test models with different numbers of hidden layers and units.

In [20]:

```python
# Define all combinations of hidden layers and units per layer
configurations = [(layers, units) for layers in range(1, 4) for units in [8, 16, 32, 64]]

# Dictionary to store results
config_results = {}

# Train models and evaluate
for layers_count, units in configurations:
    print(f"Training model with {layers_count} hidden layers and {units} units per layer...")
```

```python
    model = build_model(hidden_layers=layers_count, hidden_units=units)
    history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
                        validation_data=(x_val, y_val), verbose=0)
    eval_results = model.evaluate(x_test, y_test, verbose=0)  # [Loss, Accuracy]

    config_results[(layers_count, units)] = eval_results

# Convert results to DataFrame
config_results_df = pd.DataFrame.from_dict(config_results, orient="index", columns=["Loss", "Accuracy"])
config_results_df.index.names = ["Hidden Layers, Units"]
print(config_results_df)
```

```
Training model with 1 hidden layers and 8 units per layer...
```

```
C:\Users\zhanguu\AppData\Local\miniconda3\envs\tf_env2\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
t(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

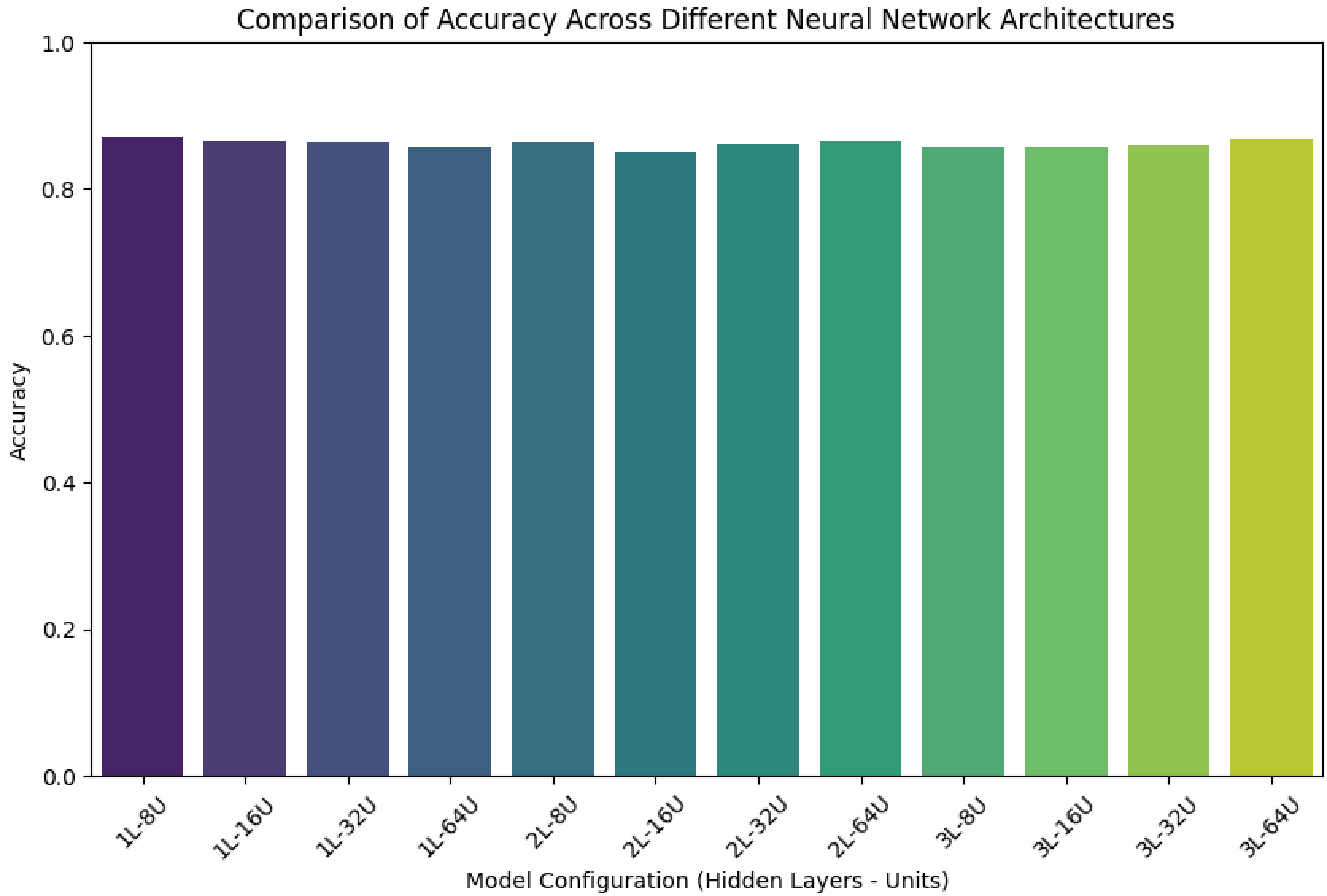
```
Training model with 1 hidden layers and 16 units per layer...
Training model with 1 hidden layers and 32 units per layer...
Training model with 1 hidden layers and 64 units per layer...
Training model with 2 hidden layers and 8 units per layer...
Training model with 2 hidden layers and 16 units per layer...
Training model with 2 hidden layers and 32 units per layer...
Training model with 2 hidden layers and 64 units per layer...
Training model with 3 hidden layers and 8 units per layer...
Training model with 3 hidden layers and 16 units per layer...
Training model with 3 hidden layers and 32 units per layer...
Training model with 3 hidden layers and 64 units per layer...
                          Loss  Accuracy
Hidden Layers, Units
(1, 8)                0.361747   0.87064
(1, 16)               0.404524   0.86548
(1, 32)               0.429601   0.86428
(1, 64)               0.490549   0.85756
(2, 8)                0.461372   0.86404
(2, 16)               0.627186   0.85000
(2, 32)               0.672112   0.86132
(2, 64)               0.634104   0.86564
(3, 8)                0.529646   0.85684
(3, 16)               0.705806   0.85688
(3, 32)               0.780766   0.85996
(3, 64)               0.763498   0.86708
```

# Step 4: Visualizing Performance of Different Architectures

In [21]:

```python
# Plot Accuracy Comparison
plt.figure(figsize=(10, 6))
sns.barplot(x=[f"{layers}L-{units}U" for layers, units in config_results.keys()],
            y=[results[1] for results in config_results.values()], palette="viridis")

plt.xlabel("Model Configuration (Hidden Layers - Units)")
plt.ylabel("Accuracy")
plt.title("Comparison of Accuracy Across Different Neural Network Architectures")
plt.xticks(rotation=45)
plt.ylim(0, 1)
plt.show()
```

```
C:\Users\zhanguu\AppData\Local\Temp\ipykernel_26204\2984978426.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `h
ue` and set `legend=False` for the same effect.

  sns.barplot(x=[f"{layers}L-{units}U" for layers, units in config_results.keys()],
```



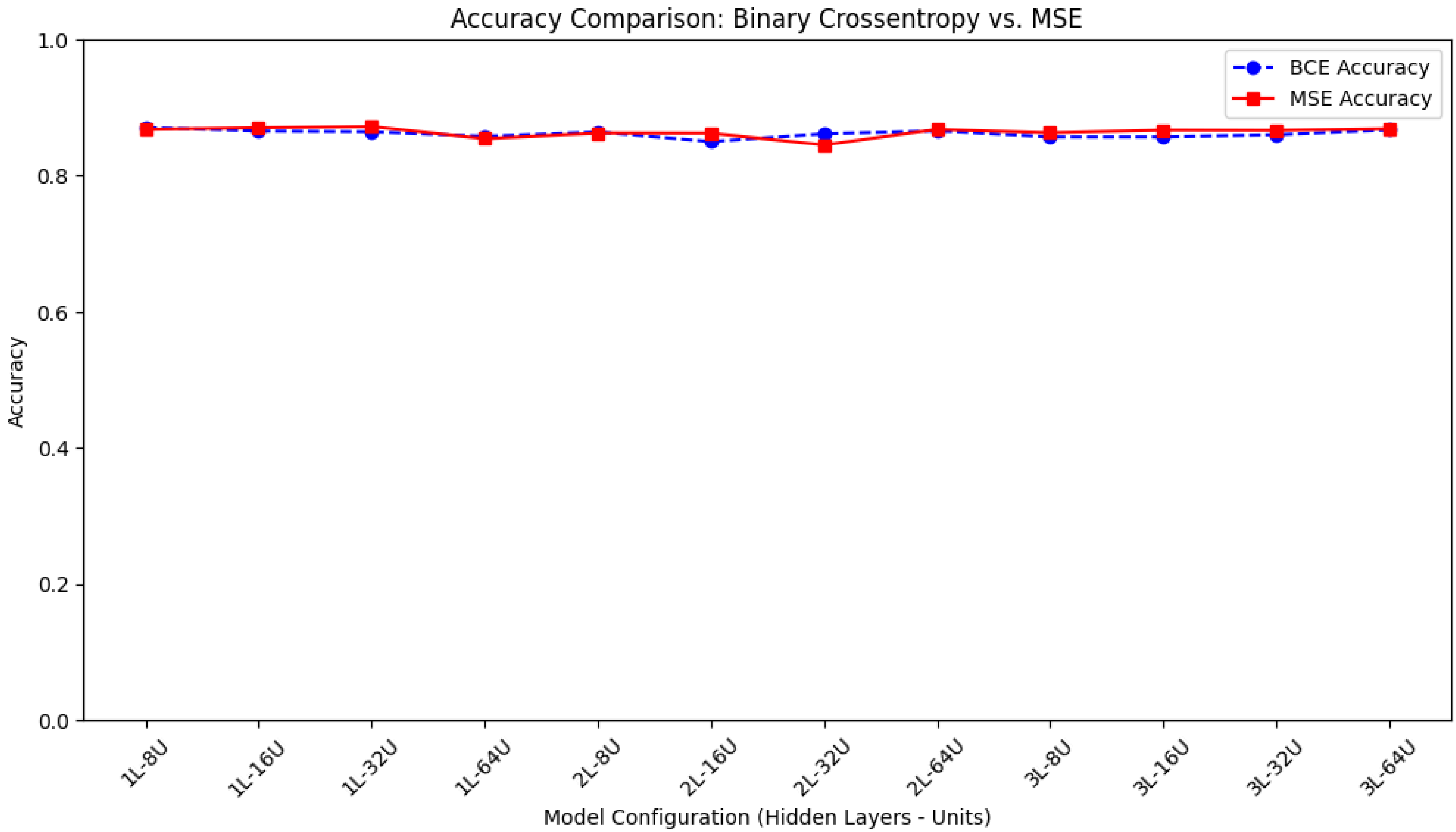
## Step 5: Changing Loss Function to MSE

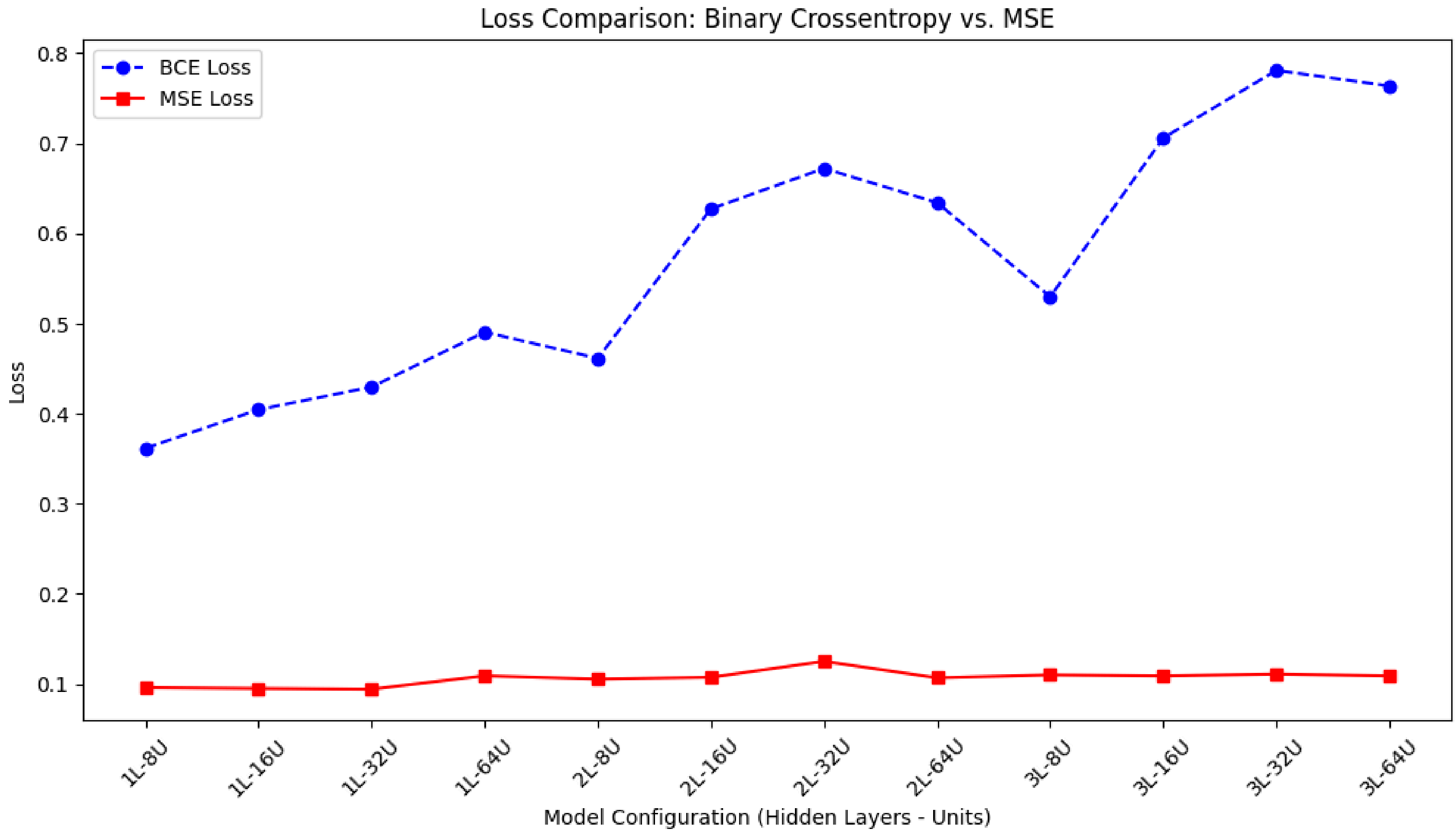
In [22]:

```
# Dictionary to store MSE results
mse_results = {}

# Train models using MSE loss function
for layers_count, units in configurations:
    print(f"Training model with {layers_count} hidden layers and {units} units per layer using MSE loss...")

    mse_model = build_model(hidden_layers=layers_count, hidden_units=units, loss_fn="mse")
    mse_history = mse_model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
                                validation_data=(x_val, y_val), verbose=0)
    eval_results = mse_model.evaluate(x_test, y_test, verbose=0)  # [Loss, Accuracy]

    mse_results[(layers_count, units)] = eval_results

# Convert MSE results to DataFrame
mse_results_df = pd.DataFrame.from_dict(mse_results, orient="index", columns=["Loss", "Accuracy"])
mse_results_df.index.names = ["Hidden Layers, Units"]

# Combine BCE and MSE results into one DataFrame
comparison_df = config_results_df.join(mse_results_df, lsuffix="_BCE", rsuffix="_MSE")
print(comparison_df)

# --- Plot Accuracy Comparison ---
plt.figure(figsize=(12, 6))
x_labels = [f"{layers}L-{units}U" for layers, units in configurations]

plt.plot(x_labels, comparison_df["Accuracy_BCE"], marker='o', label="BCE Accuracy", linestyle='--', color='blue')
plt.plot(x_labels, comparison_df["Accuracy_MSE"], marker='s', label="MSE Accuracy", linestyle='-', color='red')

plt.xlabel("Model Configuration (Hidden Layers - Units)")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison: Binary Crossentropy vs. MSE")
plt.xticks(rotation=45)
plt.legend()
plt.ylim(0, 1)
plt.show()

# --- Plot Loss Comparison ---
```

```python
plt.figure(figsize=(12, 6))
plt.plot(x_labels, comparison_df["Loss_BCE"], marker='o', label="BCE Loss", linestyle='--', color='blue')
plt.plot(x_labels, comparison_df["Loss_MSE"], marker='s', label="MSE Loss", linestyle='-', color='red')

plt.xlabel("Model Configuration (Hidden Layers - Units)")
plt.ylabel("Loss")
plt.title("Loss Comparison: Binary Crossentropy vs. MSE")
plt.xticks(rotation=45)
plt.legend()
plt.show()
```

```
Training model with 1 hidden layers and 8 units per layer using MSE loss...
C:\Users\zhanguu\AppData\Local\miniconda3\envs\tf_env2\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
t(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Training model with 1 hidden layers and 16 units per layer using MSE loss...
Training model with 1 hidden layers and 32 units per layer using MSE loss...
Training model with 1 hidden layers and 64 units per layer using MSE loss...
Training model with 2 hidden layers and 8 units per layer using MSE loss...
Training model with 2 hidden layers and 16 units per layer using MSE loss...
Training model with 2 hidden layers and 32 units per layer using MSE loss...
Training model with 2 hidden layers and 64 units per layer using MSE loss...
Training model with 3 hidden layers and 8 units per layer using MSE loss...
Training model with 3 hidden layers and 16 units per layer using MSE loss...
Training model with 3 hidden layers and 32 units per layer using MSE loss...
Training model with 3 hidden layers and 64 units per layer using MSE loss...
                      Loss_BCE  Accuracy_BCE  Loss_MSE  Accuracy_MSE
Hidden Layers, Units
(1, 8)                0.361747       0.87064  0.096216       0.86788
(1, 16)               0.404524       0.86548  0.095235       0.87036
(1, 32)               0.429601       0.86428  0.094448       0.87208
(1, 64)               0.490549       0.85756  0.109183       0.85416
(2, 8)                0.461372       0.86404  0.105588       0.86236
(2, 16)               0.627186       0.85000  0.107584       0.86216
(2, 32)               0.672112       0.86132  0.125297       0.84504
(2, 64)               0.634104       0.86564  0.106975       0.86784
(3, 8)                0.529646       0.85684  0.110227       0.86284
(3, 16)               0.705806       0.85688  0.109105       0.86684
(3, 32)               0.780766       0.85996  0.111202       0.86656
(3, 64)               0.763498       0.86708  0.109137       0.86880
```


Accuracy Comparison: Binary Crossentropy vs. MSE
BCE Accuracy
MSE Accuracy
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
1L-8U
1L-16U
1L-32U
1L-64U
2L-8U
2L-16U
2L-32U
2L-64U
3L-8U
3L-16U
3L-32U
3L-64U
Model Configuration (Hidden Layers - Units)



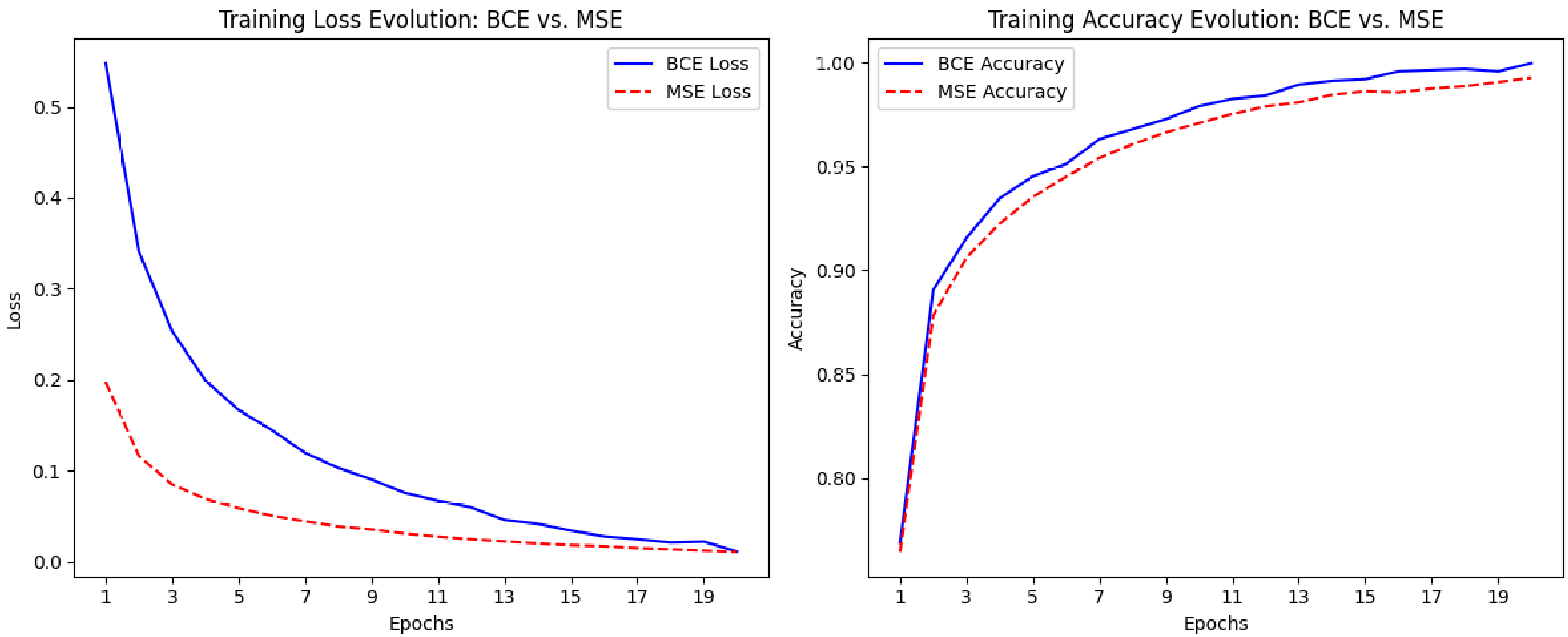

In [24]:
```
# Train a model with (2 hidden layers, 16 units) using BCE for 20 epochs
bce_model = build_model(hidden_layers=2, hidden_units=16, loss_fn="binary_crossentropy")
bce_history = bce_model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
                            validation_data=(x_val, y_val), verbose=1)

# Train a model with (2 hidden layers, 16 units) using MSE for 20 epochs
mse_model = build_model(hidden_layers=2, hidden_units=16, loss_fn="mse")
mse_history = mse_model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
                            validation_data=(x_val, y_val), verbose=1)

# Plot Training Evolution for Loss and Accuracy over all 20 Epochs
plt.figure(figsize=(12, 5))
```

```python
# Plot Loss Evolution
plt.subplot(1, 2, 1)
plt.plot(range(1, 21), bce_history.history["loss"], label="BCE Loss", color="blue")
plt.plot(range(1, 21), mse_history.history["loss"], label="MSE Loss", color="red", linestyle="--")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss Evolution: BCE vs. MSE")
plt.xticks(range(1, 21, 2))  # Show all 20 epochs clearly
plt.legend()

# Plot Accuracy Evolution
plt.subplot(1, 2, 2)
plt.plot(range(1, 21), bce_history.history["accuracy"], label="BCE Accuracy", color="blue")
plt.plot(range(1, 21), mse_history.history["accuracy"], label="MSE Accuracy", color="red", linestyle="--")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy Evolution: BCE vs. MSE")
plt.xticks(range(1, 21, 2))  # Show all 20 epochs clearly
plt.legend()

plt.tight_layout()
plt.show()
```

```
C:\Users\zhanguu\AppData\Local\miniconda3\envs\tf_env2\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
t(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.6789 - loss: 0.6170 - val_accuracy: 0.8665 - val_loss: 0.4149
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8894 - loss: 0.3624 - val_accuracy: 0.8811 - val_loss: 0.3260
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9173 - loss: 0.2575 - val_accuracy: 0.8901 - val_loss: 0.2857
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9347 - loss: 0.2017 - val_accuracy: 0.8743 - val_loss: 0.3120
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9441 - loss: 0.1706 - val_accuracy: 0.8806 - val_loss: 0.2992
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9528 - loss: 0.1432 - val_accuracy: 0.8855 - val_loss: 0.2869
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9651 - loss: 0.1181 - val_accuracy: 0.8849 - val_loss: 0.2950
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9719 - loss: 0.0982 - val_accuracy: 0.8825 - val_loss: 0.3187
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9740 - loss: 0.0876 - val_accuracy: 0.8820 - val_loss: 0.3343
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9791 - loss: 0.0752 - val_accuracy: 0.8747 - val_loss: 0.3821
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9842 - loss: 0.0657 - val_accuracy: 0.8817 - val_loss: 0.3622
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9872 - loss: 0.0546 - val_accuracy: 0.8787 - val_loss: 0.3831
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9922 - loss: 0.0419 - val_accuracy: 0.8767 - val_loss: 0.4007
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9940 - loss: 0.0363 - val_accuracy: 0.8721 - val_loss: 0.4465
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9948 - loss: 0.0299 - val_accuracy: 0.8733 - val_loss: 0.4469
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9967 - loss: 0.0259 - val_accuracy: 0.8611 - val_loss: 0.5083
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9980 - loss: 0.0214 - val_accuracy: 0.8701 - val_loss: 0.4964
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9981 - loss: 0.0179 - val_accuracy: 0.8739 - val_loss: 0.5164
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9981 - loss: 0.0162 - val_accuracy: 0.8719 - val_loss: 0.5349
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9996 - loss: 0.0103 - val_accuracy: 0.8715 - val_loss: 0.5610
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.6900 - loss: 0.2225 - val_accuracy: 0.8460 - val_loss: 0.1460
```

```
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8730 - loss: 0.1249 - val_accuracy: 0.8688 - val_loss: 0.1087
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9041 - loss: 0.0889 - val_accuracy: 0.8721 - val_loss: 0.0981
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9187 - loss: 0.0719 - val_accuracy: 0.8880 - val_loss: 0.0870
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9369 - loss: 0.0593 - val_accuracy: 0.8865 - val_loss: 0.0849
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9481 - loss: 0.0493 - val_accuracy: 0.8874 - val_loss: 0.0841
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9582 - loss: 0.0431 - val_accuracy: 0.8741 - val_loss: 0.0926
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9609 - loss: 0.0385 - val_accuracy: 0.8855 - val_loss: 0.0832
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9665 - loss: 0.0350 - val_accuracy: 0.8831 - val_loss: 0.0844
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9733 - loss: 0.0297 - val_accuracy: 0.8718 - val_loss: 0.0965
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9755 - loss: 0.0279 - val_accuracy: 0.8797 - val_loss: 0.0882
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9810 - loss: 0.0235 - val_accuracy: 0.8792 - val_loss: 0.0875
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9816 - loss: 0.0219 - val_accuracy: 0.8716 - val_loss: 0.0944
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9846 - loss: 0.0202 - val_accuracy: 0.8784 - val_loss: 0.0903
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9854 - loss: 0.0184 - val_accuracy: 0.8756 - val_loss: 0.0918
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9870 - loss: 0.0161 - val_accuracy: 0.8745 - val_loss: 0.0931
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9895 - loss: 0.0134 - val_accuracy: 0.8760 - val_loss: 0.0937
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9917 - loss: 0.0114 - val_accuracy: 0.8739 - val_loss: 0.0954
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9921 - loss: 0.0104 - val_accuracy: 0.8749 - val_loss: 0.0963
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9929 - loss: 0.0097 - val_accuracy: 0.8754 - val_loss: 0.0972
```




## Step 6: Experimenting with Activation Functions (ReLU vs. Tanh)

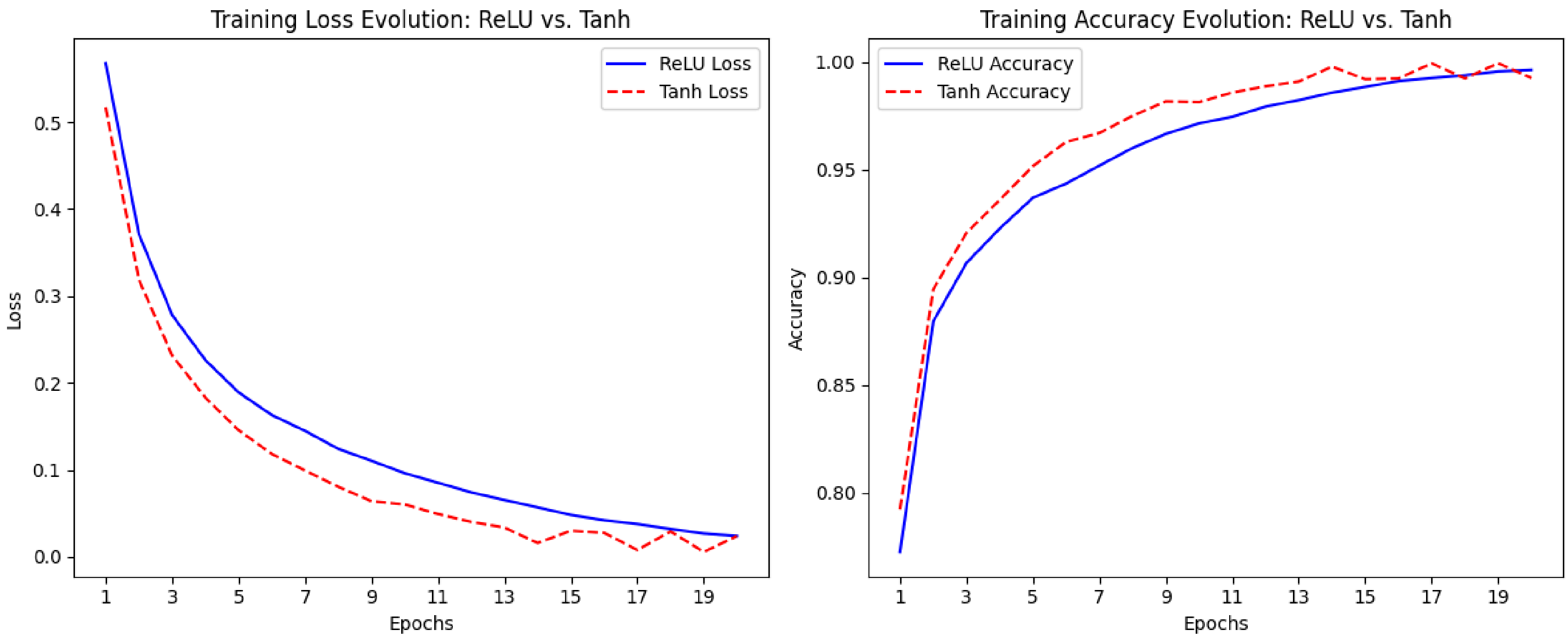
In [25]:
```
# Train a model with ReLU activation
relu_model = build_model(activation="relu")
relu_history = relu_model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
                              validation_data=(x_val, y_val), verbose=1)

# Train a model with Tanh activation
tanh_model = build_model(activation="tanh")
tanh_history = tanh_model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
                              validation_data=(x_val, y_val), verbose=1)

# Plot Training Evolution for Loss and Accuracy over 20 Epochs
plt.figure(figsize=(12, 5))

# Plot Loss Evolution
plt.subplot(1, 2, 1)
plt.plot(range(1, 21), relu_history.history["loss"], label="ReLU Loss", color="blue")
plt.plot(range(1, 21), tanh_history.history["loss"], label="Tanh Loss", color="red", linestyle="--")
```

```python
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss Evolution: ReLU vs. Tanh")
plt.xticks(range(1, 21, 2))  # Show all 20 epochs clearly
plt.legend()

# Plot Accuracy Evolution
plt.subplot(1, 2, 2)
plt.plot(range(1, 21), relu_history.history["accuracy"], label="ReLU Accuracy", color="blue")
plt.plot(range(1, 21), tanh_history.history["accuracy"], label="Tanh Accuracy", color="red", linestyle="--")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy Evolution: ReLU vs. Tanh")
plt.xticks(range(1, 21, 2))  # Show all 20 epochs clearly
plt.legend()

plt.tight_layout()
plt.show()
```

```
C:\Users\zhanguu\AppData\Local\miniconda3\envs\tf_env2\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
t(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.6894 - loss: 0.6287 - val_accuracy: 0.8039 - val_loss: 0.4755
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8701 - loss: 0.3985 - val_accuracy: 0.8772 - val_loss: 0.3408
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9080 - loss: 0.2854 - val_accuracy: 0.8717 - val_loss: 0.3165
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9222 - loss: 0.2302 - val_accuracy: 0.8855 - val_loss: 0.2863
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9391 - loss: 0.1871 - val_accuracy: 0.8876 - val_loss: 0.2786
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9429 - loss: 0.1645 - val_accuracy: 0.8862 - val_loss: 0.2787
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9548 - loss: 0.1423 - val_accuracy: 0.8831 - val_loss: 0.2910
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9615 - loss: 0.1237 - val_accuracy: 0.8831 - val_loss: 0.3089
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9685 - loss: 0.1108 - val_accuracy: 0.8792 - val_loss: 0.3147
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9738 - loss: 0.0932 - val_accuracy: 0.8749 - val_loss: 0.3396
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9778 - loss: 0.0800 - val_accuracy: 0.8800 - val_loss: 0.3352
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9809 - loss: 0.0707 - val_accuracy: 0.8744 - val_loss: 0.3673
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9849 - loss: 0.0618 - val_accuracy: 0.8704 - val_loss: 0.3893
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9882 - loss: 0.0544 - val_accuracy: 0.8719 - val_loss: 0.4018
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9912 - loss: 0.0450 - val_accuracy: 0.8737 - val_loss: 0.4167
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9939 - loss: 0.0381 - val_accuracy: 0.8711 - val_loss: 0.4426
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9938 - loss: 0.0348 - val_accuracy: 0.8725 - val_loss: 0.4705
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9961 - loss: 0.0288 - val_accuracy: 0.8711 - val_loss: 0.4929
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9960 - loss: 0.0260 - val_accuracy: 0.8693 - val_loss: 0.5047
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9976 - loss: 0.0218 - val_accuracy: 0.8676 - val_loss: 0.5350
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.7076 - loss: 0.5911 - val_accuracy: 0.8407 - val_loss: 0.4154
```

```
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8932 - loss: 0.3358 - val_accuracy: 0.8717 - val_loss: 0.3223
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9240 - loss: 0.2352 - val_accuracy: 0.8807 - val_loss: 0.2877
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9433 - loss: 0.1772 - val_accuracy: 0.8860 - val_loss: 0.2726
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9528 - loss: 0.1462 - val_accuracy: 0.8845 - val_loss: 0.2845
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9689 - loss: 0.1064 - val_accuracy: 0.8832 - val_loss: 0.3049
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9729 - loss: 0.0926 - val_accuracy: 0.8822 - val_loss: 0.3292
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9801 - loss: 0.0721 - val_accuracy: 0.8664 - val_loss: 0.3985
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9845 - loss: 0.0595 - val_accuracy: 0.8554 - val_loss: 0.4753
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9823 - loss: 0.0617 - val_accuracy: 0.8688 - val_loss: 0.4753
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9911 - loss: 0.0382 - val_accuracy: 0.8710 - val_loss: 0.4579
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9946 - loss: 0.0291 - val_accuracy: 0.8666 - val_loss: 0.4963
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9961 - loss: 0.0233 - val_accuracy: 0.8654 - val_loss: 0.5313
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9981 - loss: 0.0155 - val_accuracy: 0.8660 - val_loss: 0.5560
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9933 - loss: 0.0278 - val_accuracy: 0.8674 - val_loss: 0.5796
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9973 - loss: 0.0137 - val_accuracy: 0.8648 - val_loss: 0.6032
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9995 - loss: 0.0087 - val_accuracy: 0.8634 - val_loss: 0.6358
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9932 - loss: 0.0279 - val_accuracy: 0.8660 - val_loss: 0.6406
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9995 - loss: 0.0055 - val_accuracy: 0.8647 - val_loss: 0.6682
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9965 - loss: 0.0131 - val_accuracy: 0.8650 - val_loss: 0.6811
```




## Step 7: Applying Regularization Techniques

In [26]:

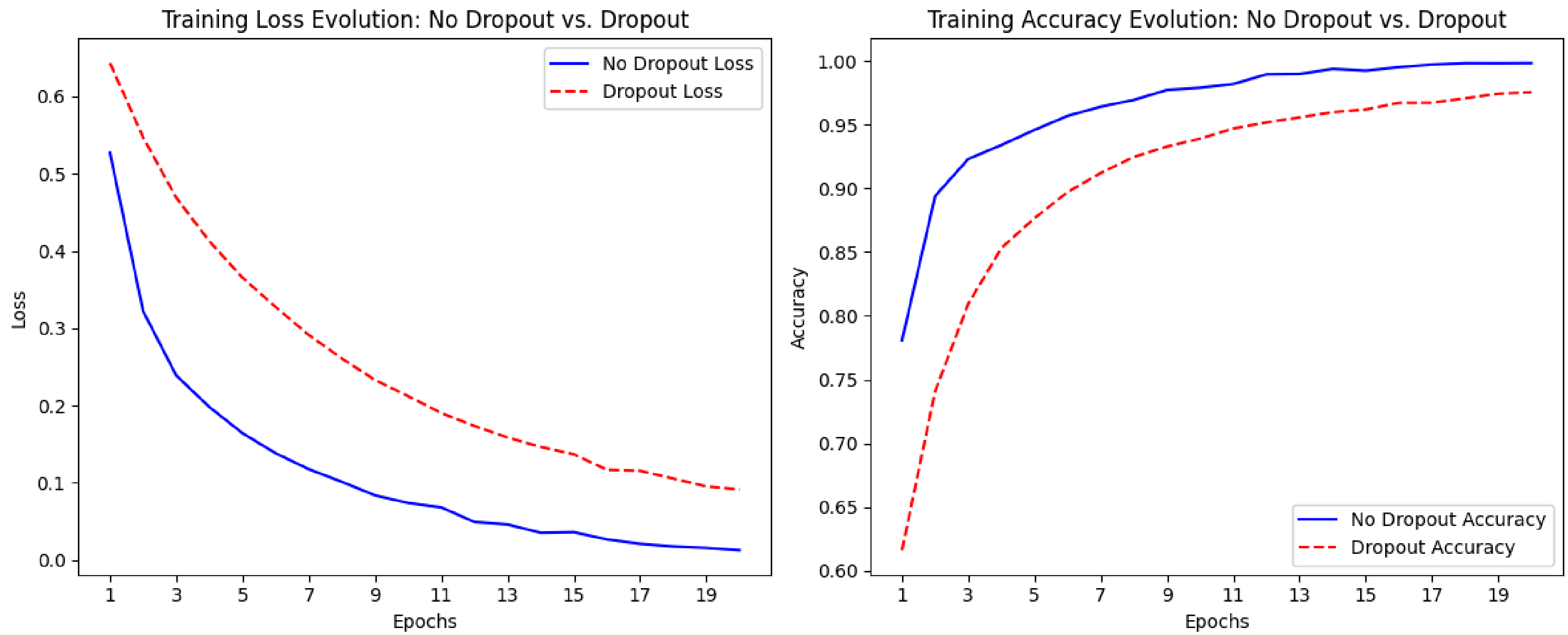
```
# Train a model WITHOUT Dropout (Standard Model)
no_dropout_model = build_model()
no_dropout_history = no_dropout_model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
                                          validation_data=(x_val, y_val), verbose=1)

# Train a model WITH Dropout Regularization
dropout_model = build_model_with_dropout()
dropout_history = dropout_model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
                                    validation_data=(x_val, y_val), verbose=1)

# Plot Training Evolution for Loss and Accuracy over 20 Epochs
plt.figure(figsize=(12, 5))

# Plot Loss Evolution
plt.subplot(1, 2, 1)
plt.plot(range(1, 21), no_dropout_history.history["loss"], label="No Dropout Loss", color="blue")
plt.plot(range(1, 21), dropout_history.history["loss"], label="Dropout Loss", color="red", linestyle="--")
```

```python
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss Evolution: No Dropout vs. Dropout")
plt.xticks(range(1, 21, 2))  # Show all 20 epochs clearly
plt.legend()

# Plot Accuracy Evolution
plt.subplot(1, 2, 2)
plt.plot(range(1, 21), no_dropout_history.history["accuracy"], label="No Dropout Accuracy", color="blue")
plt.plot(range(1, 21), dropout_history.history["accuracy"], label="Dropout Accuracy", color="red", linestyle="--")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy Evolution: No Dropout vs. Dropout")
plt.xticks(range(1, 21, 2))  # Show all 20 epochs clearly
plt.legend()

plt.tight_layout()
plt.show()
```

```
C:\Users\zhanguu\AppData\Local\miniconda3\envs\tf_env2\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
t(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.6978 - loss: 0.6022 - val_accuracy: 0.8643 - val_loss: 0.3925
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8908 - loss: 0.3397 - val_accuracy: 0.8823 - val_loss: 0.3155
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9250 - loss: 0.2436 - val_accuracy: 0.8402 - val_loss: 0.3687
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9297 - loss: 0.2079 - val_accuracy: 0.8816 - val_loss: 0.2921
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9495 - loss: 0.1604 - val_accuracy: 0.8885 - val_loss: 0.2762
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9626 - loss: 0.1306 - val_accuracy: 0.8838 - val_loss: 0.2897
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9664 - loss: 0.1146 - val_accuracy: 0.8735 - val_loss: 0.3235
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9730 - loss: 0.0971 - val_accuracy: 0.8832 - val_loss: 0.3063
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9789 - loss: 0.0793 - val_accuracy: 0.8809 - val_loss: 0.3400
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9801 - loss: 0.0728 - val_accuracy: 0.8808 - val_loss: 0.3563
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9853 - loss: 0.0616 - val_accuracy: 0.8816 - val_loss: 0.3560
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9920 - loss: 0.0459 - val_accuracy: 0.8774 - val_loss: 0.3770
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9929 - loss: 0.0412 - val_accuracy: 0.8785 - val_loss: 0.4008
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9947 - loss: 0.0338 - val_accuracy: 0.8659 - val_loss: 0.4552
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9925 - loss: 0.0361 - val_accuracy: 0.8763 - val_loss: 0.4420
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9965 - loss: 0.0225 - val_accuracy: 0.8747 - val_loss: 0.4636
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9980 - loss: 0.0183 - val_accuracy: 0.8752 - val_loss: 0.4866
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9991 - loss: 0.0158 - val_accuracy: 0.8733 - val_loss: 0.5126
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9992 - loss: 0.0132 - val_accuracy: 0.8723 - val_loss: 0.5402
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9987 - loss: 0.0116 - val_accuracy: 0.8696 - val_loss: 0.5681
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.5702 - loss: 0.6688 - val_accuracy: 0.8274 - val_loss: 0.5467
```

```
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.7239 - loss: 0.5633 - val_accuracy: 0.8641 - val_loss: 0.4496
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8023 - loss: 0.4779 - val_accuracy: 0.8772 - val_loss: 0.3844
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8464 - loss: 0.4237 - val_accuracy: 0.8822 - val_loss: 0.3407
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8760 - loss: 0.3694 - val_accuracy: 0.8830 - val_loss: 0.3142
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8953 - loss: 0.3305 - val_accuracy: 0.8899 - val_loss: 0.2872
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9132 - loss: 0.2916 - val_accuracy: 0.8819 - val_loss: 0.3085
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9253 - loss: 0.2621 - val_accuracy: 0.8849 - val_loss: 0.3013
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9348 - loss: 0.2313 - val_accuracy: 0.8870 - val_loss: 0.2892
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9400 - loss: 0.2109 - val_accuracy: 0.8893 - val_loss: 0.2943
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9486 - loss: 0.1885 - val_accuracy: 0.8874 - val_loss: 0.2964
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9502 - loss: 0.1778 - val_accuracy: 0.8878 - val_loss: 0.3162
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9566 - loss: 0.1588 - val_accuracy: 0.8870 - val_loss: 0.3287
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9600 - loss: 0.1490 - val_accuracy: 0.8855 - val_loss: 0.3464
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9605 - loss: 0.1416 - val_accuracy: 0.8852 - val_loss: 0.3785
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9673 - loss: 0.1150 - val_accuracy: 0.8837 - val_loss: 0.4131
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9687 - loss: 0.1100 - val_accuracy: 0.8824 - val_loss: 0.4121
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9691 - loss: 0.1045 - val_accuracy: 0.8826 - val_loss: 0.4372
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9759 - loss: 0.0939 - val_accuracy: 0.8824 - val_loss: 0.4618
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9755 - loss: 0.0927 - val_accuracy: 0.8789 - val_loss: 0.5138
```




# Step 8: Dropout and L2 Regularization

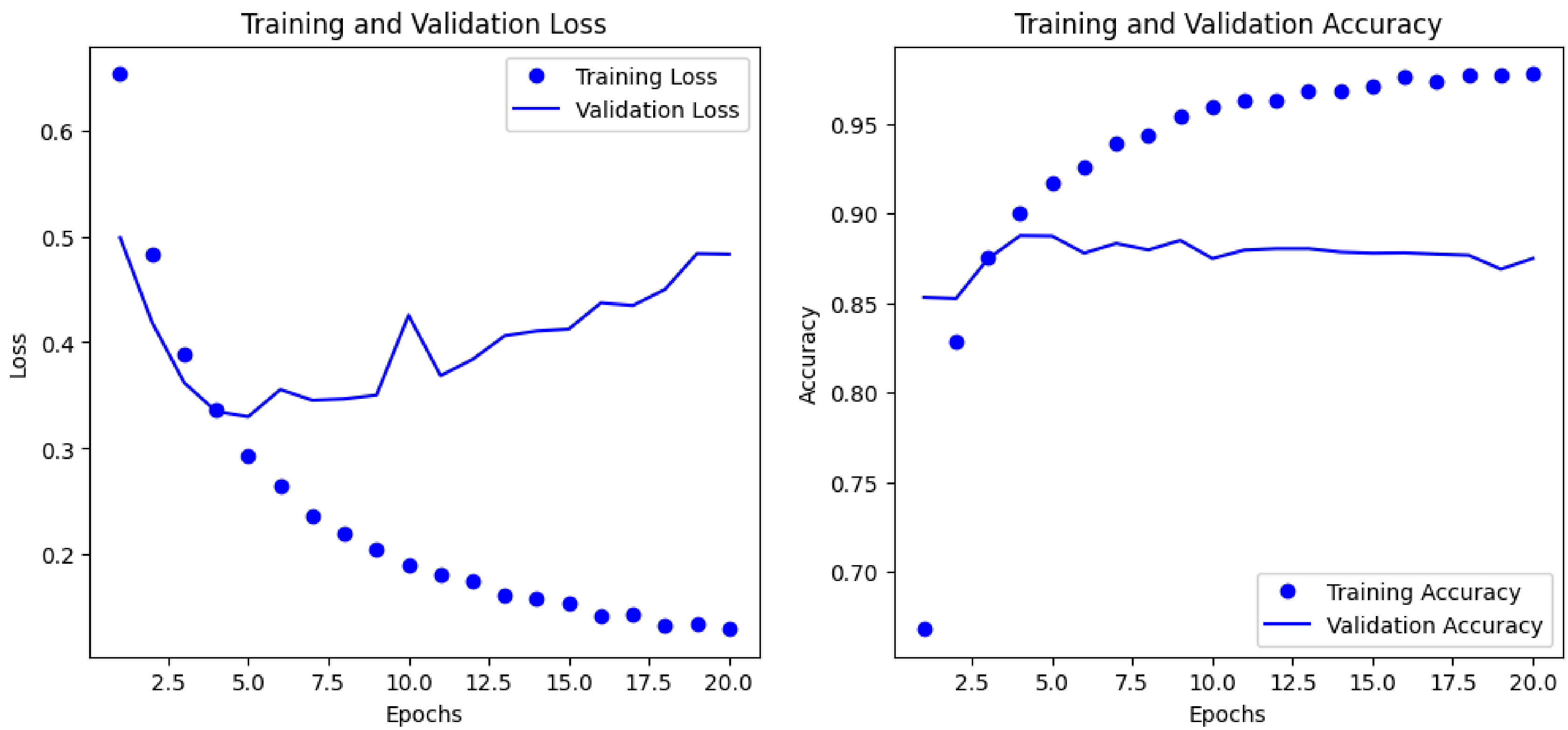
In [27]:

```
# Train a model with Dropout and L2 Regularization
def build_model_with_dropout_and_l2():
    model = keras.Sequential([
        layers.Dense(16, activation="relu", kernel_regularizer=keras.regularizers.l2(0.001), input_shape=(10000,)),
        layers.Dropout(0.3),
        layers.Dense(16, activation="relu", kernel_regularizer=keras.regularizers.l2(0.001)),
        layers.Dropout(0.3),
        layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
    return model

dropout_l2_model = build_model_with_dropout_and_l2()
dropout_l2_history = dropout_l2_model.fit(
    partial_x_train, partial_y_train, epochs=20, batch_size=512,
    validation_data=(x_val, y_val), verbose=1
)
```

```python
# Plot training and validation loss
import matplotlib.pyplot as plt

history = dropout_l2_history.history

epochs = range(1, 21)
plt.figure(figsize=(12, 5))

# Plot loss
plt.subplot(1, 2, 1)
plt.plot(epochs, history["loss"], "bo", label="Training Loss")
plt.plot(epochs, history["val_loss"], "b", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

# Plot accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, history["accuracy"], "bo", label="Training Accuracy")
plt.plot(epochs, history["val_accuracy"], "b", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.show()
```

```
C:\Users\zhanguu\AppData\Local\miniconda3\envs\tf_env2\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
t(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.5880 - loss: 0.7011 - val_accuracy: 0.8533 - val_loss: 0.4986
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8141 - loss: 0.5039 - val_accuracy: 0.8526 - val_loss: 0.4188
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8729 - loss: 0.3959 - val_accuracy: 0.8748 - val_loss: 0.3620
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8969 - loss: 0.3411 - val_accuracy: 0.8877 - val_loss: 0.3348
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9157 - loss: 0.2953 - val_accuracy: 0.8875 - val_loss: 0.3299
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9256 - loss: 0.2660 - val_accuracy: 0.8780 - val_loss: 0.3555
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9439 - loss: 0.2326 - val_accuracy: 0.8834 - val_loss: 0.3454
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9497 - loss: 0.2152 - val_accuracy: 0.8798 - val_loss: 0.3467
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9580 - loss: 0.1998 - val_accuracy: 0.8851 - val_loss: 0.3503
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9612 - loss: 0.1877 - val_accuracy: 0.8750 - val_loss: 0.4256
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9642 - loss: 0.1789 - val_accuracy: 0.8797 - val_loss: 0.3686
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9678 - loss: 0.1679 - val_accuracy: 0.8805 - val_loss: 0.3839
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9735 - loss: 0.1552 - val_accuracy: 0.8805 - val_loss: 0.4061
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9731 - loss: 0.1512 - val_accuracy: 0.8786 - val_loss: 0.4107
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9740 - loss: 0.1488 - val_accuracy: 0.8780 - val_loss: 0.4125
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9788 - loss: 0.1396 - val_accuracy: 0.8782 - val_loss: 0.4371
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9782 - loss: 0.1338 - val_accuracy: 0.8774 - val_loss: 0.4347
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9818 - loss: 0.1280 - val_accuracy: 0.8768 - val_loss: 0.4498
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9814 - loss: 0.1250 - val_accuracy: 0.8690 - val_loss: 0.4836
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9795 - loss: 0.1263 - val_accuracy: 0.8750 - val_loss: 0.4830
```



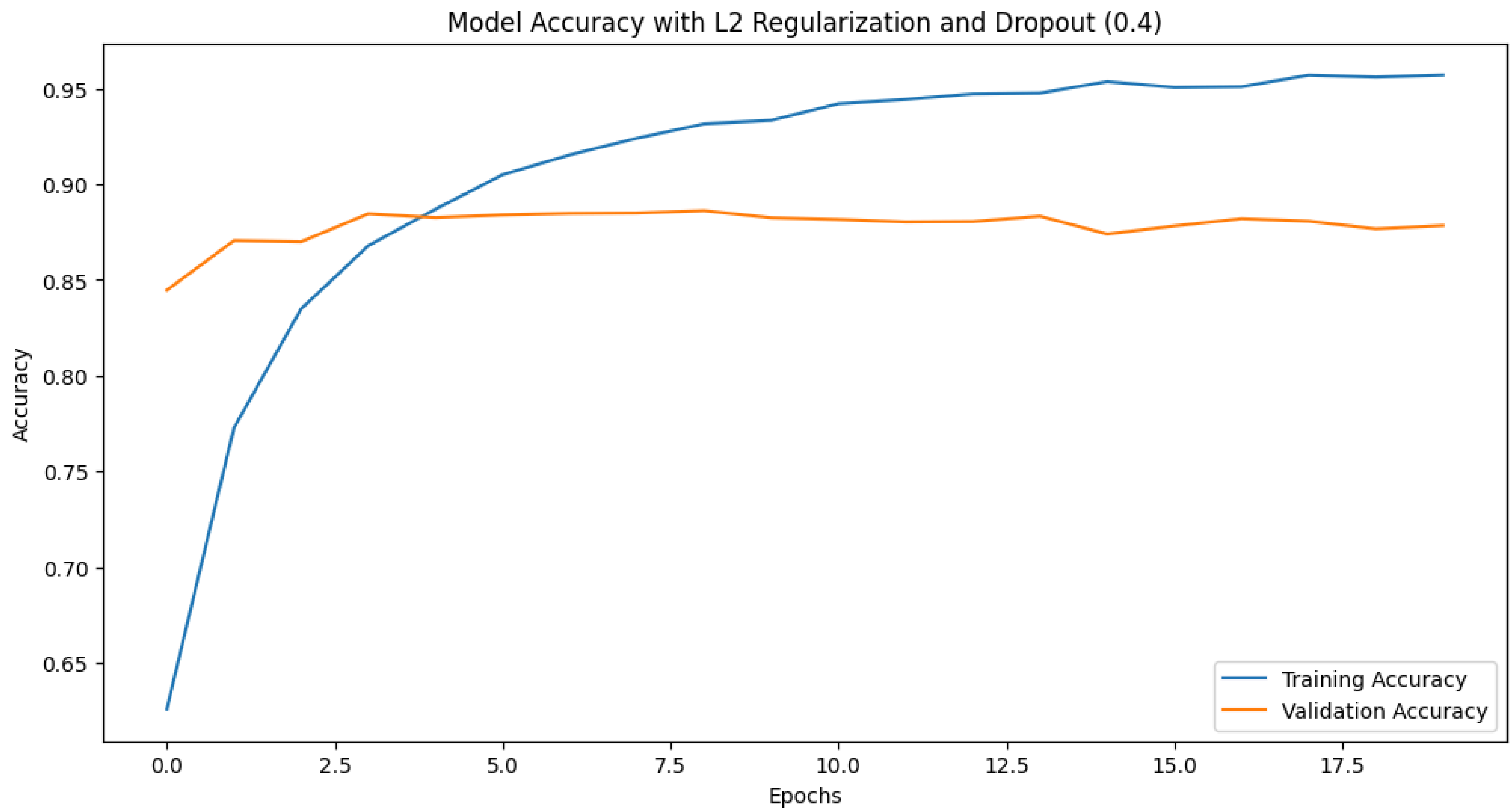

In [28]:

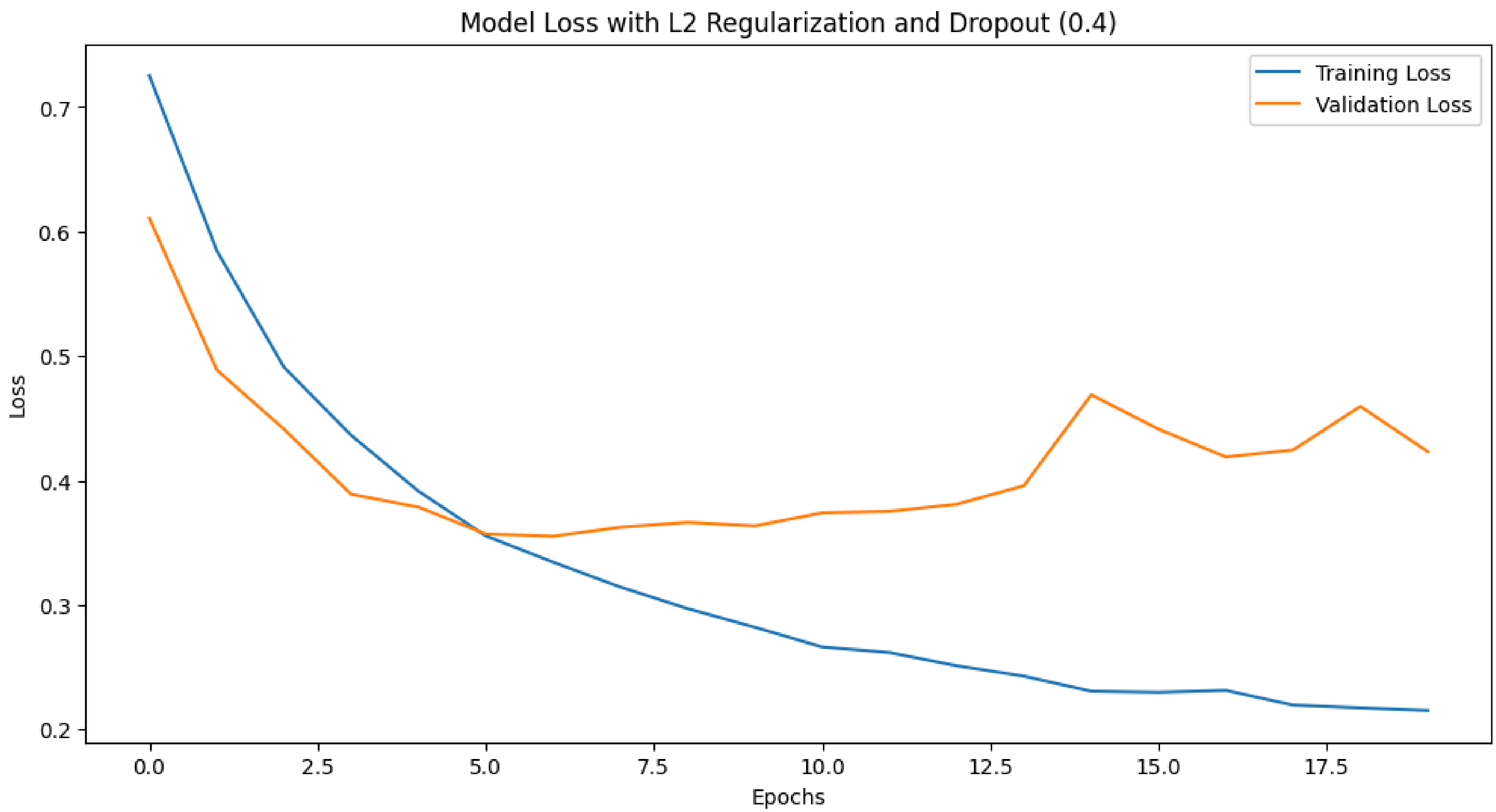
```
# Train a model with L2 regularization and adjusted dropout
def build_model_with_l2_and_dropout():
    model = keras.Sequential([
        layers.Dense(16, activation="relu", input_shape=(10000,), kernel_regularizer=keras.regularizers.l2(0.002)),
        layers.Dropout(0.4),
        layers.Dense(16, activation="relu", kernel_regularizer=keras.regularizers.l2(0.002)),
        layers.Dropout(0.4),
        layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer="rmsprop", loss="binary_crossentropy", metrics=["accuracy"])
    return model

# Train the model with L2 regularization and adjusted dropout
model_with_l2_and_dropout = build_model_with_l2_and_dropout()
history_with_l2_and_dropout = model_with_l2_and_dropout.fit(partial_x_train, partial_y_train, epochs=20, batch_size=5
                                                            validation_data=(x_val, y_val), verbose=1)
```

```python
# Plot the results
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.figure(figsize=(12, 6))
plt.plot(history_with_l2_and_dropout.history['accuracy'], label='Training Accuracy')
plt.plot(history_with_l2_and_dropout.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy with L2 Regularization and Dropout (0.4)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')

# Plot training & validation loss values
plt.figure(figsize=(12, 6))
plt.plot(history_with_l2_and_dropout.history['loss'], label='Training Loss')
plt.plot(history_with_l2_and_dropout.history['val_loss'], label='Validation Loss')
plt.title('Model Loss with L2 Regularization and Dropout (0.4)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='upper right')

plt.show()
```

```
C:\Users\zhanguu\AppData\Local\miniconda3\envs\tf_env2\lib\site-packages\keras\src\layers\core\dense.py:87: UserWarni
ng: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inpu
t(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.5770 - loss: 0.7590 - val_accuracy: 0.8447 - val_loss: 0.6106
Epoch 2/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.7599 - loss: 0.6051 - val_accuracy: 0.8706 - val_loss: 0.4889
Epoch 3/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8247 - loss: 0.5051 - val_accuracy: 0.8700 - val_loss: 0.4412
Epoch 4/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8653 - loss: 0.4446 - val_accuracy: 0.8846 - val_loss: 0.3889
Epoch 5/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.8885 - loss: 0.3945 - val_accuracy: 0.8826 - val_loss: 0.3786
Epoch 6/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9019 - loss: 0.3584 - val_accuracy: 0.8840 - val_loss: 0.3571
Epoch 7/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9201 - loss: 0.3298 - val_accuracy: 0.8848 - val_loss: 0.3553
Epoch 8/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9274 - loss: 0.3123 - val_accuracy: 0.8850 - val_loss: 0.3624
Epoch 9/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9317 - loss: 0.2969 - val_accuracy: 0.8862 - val_loss: 0.3663
Epoch 10/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9339 - loss: 0.2799 - val_accuracy: 0.8825 - val_loss: 0.3635
Epoch 11/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9432 - loss: 0.2648 - val_accuracy: 0.8816 - val_loss: 0.3740
Epoch 12/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9441 - loss: 0.2585 - val_accuracy: 0.8803 - val_loss: 0.3752
Epoch 13/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9516 - loss: 0.2449 - val_accuracy: 0.8805 - val_loss: 0.3809
Epoch 14/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9522 - loss: 0.2322 - val_accuracy: 0.8833 - val_loss: 0.3959
Epoch 15/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9571 - loss: 0.2239 - val_accuracy: 0.8740 - val_loss: 0.4689
Epoch 16/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9514 - loss: 0.2275 - val_accuracy: 0.8782 - val_loss: 0.4412
Epoch 17/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9554 - loss: 0.2233 - val_accuracy: 0.8820 - val_loss: 0.4189
Epoch 18/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9612 - loss: 0.2091 - val_accuracy: 0.8807 - val_loss: 0.4245
Epoch 19/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9605 - loss: 0.2096 - val_accuracy: 0.8767 - val_loss: 0.4595
Epoch 20/20
30/30 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9612 - loss: 0.2081 - val_accuracy: 0.8784 - val_loss: 0.4233
```


Model Accuracy with L2 Regularization and Dropout (0.4)
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs
Training Accuracy
Validation Accuracy


Model Loss with L2 Regularization and Dropout (0.4)
Training Loss
Validation Loss
Loss
Epochs
0.2
0.3
0.4
0.5
0.6
0.7
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

高速下载