In [2]:
```
from scipy.io import arff
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the OBS-Network-DataSet_2_Aug27 file
data, meta = arff.loadarff('OBS-Network-DataSet_2_Aug27.arff')

# Convert data to pandas DataFrame
data = pd.DataFrame(data)

# Display the DataFrame
print(data.head())
```

```
   Node  Utilised Bandwith Rate  Packet Drop Rate  Full_Bandwidth  \
0   3.0                0.822038          0.190381          1000.0
1   9.0                0.275513          0.729111           100.0
2   3.0                0.923707          0.090383           900.0
3   9.0                0.368775          0.637710           100.0
4   3.0                0.905217          0.108670           800.0

   Average_Delay_Time_Per_Sec  Percentage_Of_Lost_Pcaket_Rate  \
0                    0.004815                       19.031487
1                    0.004815                       72.889036
2                    0.000633                        9.035834
3                    0.000552                       63.737843
4                    0.000497                       10.864208

   Percentage_Of_Lost_Byte_Rate  Packet Received  Rate  of Used_Bandwidth  \
0                     19.038129               0.809619          822.03750
1                     72.911141               0.270889           27.55125
2                      9.038339               0.909617          831.33600
3                     63.770999               0.362290           36.87750
4                     10.866977               0.891330          724.17375

   Lost_Bandwidth  ...  Packet_Received  Packet_lost  Transmitted_Byte  \
0       177.96250  ...          73128.0      17196.0       130066560.0
1        72.44875  ...           2451.0       6598.0        13029120.0
2        68.66400  ...          73930.0       7346.0       117037440.0
3        63.12250  ...           3278.0       5770.0        13029120.0
4        75.82625  ...          64379.0       7849.0       104008320.0

   Received_Byte  10-Run-AVG-Drop-Rate  10-Run-AVG-Bandwith-Use  10-Run-Delay  \
0    105304320.0              0.146594                 0.780936      0.001838
1      3529440.0              0.517669                 0.242451      0.002236
2    106459200.0              0.058749                 0.886758      0.001751
3      4720320.0              0.522922                 0.324522      0.001776
4     92705760.0              0.076069                 0.869009      0.001767

   Node Status  Flood Status           Class
0         b'B'      0.023455  b'NB-No Block'
1        b'NB'      0.460725         b'Block'
2         b'B'      0.000000      b'No Block'
3        b'NB'      0.439255         b'Block'
4         b'B'      0.000000      b'No Block'

[5 rows x 22 columns]
```

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1075 entries, 0 to 1074
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Node                            1075 non-null   float64
 1   Utilised Bandwith Rate          1075 non-null   float64
 2   Packet Drop Rate                1075 non-null   float64
 3   Full_Bandwidth                  1075 non-null   float64
 4   Average_Delay_Time_Per_Sec      1075 non-null   float64
 5   Percentage_Of_Lost_Pcaket_Rate  1075 non-null   float64
 6   Percentage_Of_Lost_Byte_Rate    1075 non-null   float64
 7   Packet Received  Rate           1075 non-null   float64
 8   of Used_Bandwidth               1075 non-null   float64
 9   Lost_Bandwidth                  1075 non-null   float64
 10  Packet Size_Byte                1075 non-null   float64
 11  Packet_Transmitted              1075 non-null   float64
 12  Packet_Received                 1075 non-null   float64
 13  Packet_lost                     1060 non-null   float64
 14  Transmitted_Byte                1075 non-null   float64
 15  Received_Byte                   1075 non-null   float64
 16  10-Run-AVG-Drop-Rate            1075 non-null   float64
 17  10-Run-AVG-Bandwith-Use         1075 non-null   float64
 18  10-Run-Delay                    1075 non-null   float64
 19  Node Status                     1075 non-null   object 
 20  Flood Status                    1075 non-null   float64
 21  Class                           1075 non-null   object 
dtypes: float64(20), object(2)
memory usage: 184.9+ KB
```

In [4]:
```
# Handle missing values
data.dropna(inplace=True)  # Drop rows with missing values
```

In [5]:
```
# Encode categorical variables
data = pd.get_dummies(data, columns=['Node Status', 'Flood Status'])
```

In [6]:
```
# Select numerical columns
numerical_cols = data.select_dtypes(include=['float64']).columns

# Scale numerical features
scaler = StandardScaler()
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

# Final scaled dataset
print(data.head())
```

```
       Node  Utilised Bandwith Rate  Packet Drop Rate  Full_Bandwidth  \
0 -1.009479                1.203808         -1.207882        1.595133
1  0.990610               -1.757009          1.762096       -1.536054
2 -1.009479                1.754604         -1.759164        1.247223
3  0.990610               -1.251759          1.258209       -1.536054
4 -1.009479                1.654433         -1.658349        0.899313

   Average_Delay_Time_Per_Sec  Percentage_Of_Lost_Pcaket_Rate  \
0                    4.051933                       -1.208468
1                    4.051933                        1.757715
2                   -0.319842                       -1.758974
3                   -0.404517                        1.253716
4                   -0.462013                       -1.658278

   Percentage_Of_Lost_Byte_Rate  Packet Received  Rate  of Used_Bandwidth  \
0                     -1.208093               1.208112           2.094188
1                      1.754748              -1.754692          -1.365866
2                     -1.758049               1.758062           2.134683
3                      1.252070              -1.252022          -1.325250
4                     -1.657480               1.657491           1.667983

   Lost_Bandwidth  ...  Flood Status_0.448798  Flood Status_0.457467  \
0       -0.157763  ...                  False                  False
1       -0.901891  ...                  False                  False
2       -0.928583  ...                  False                  False
3       -0.967664  ...                  False                  False
4       -0.878072  ...                  False                  False

   Flood Status_0.460725  Flood Status_0.465321  Flood Status_0.468948  \
0                  False                  False                  False
1                   True                  False                  False
2                  False                  False                  False
3                  False                  False                  False
4                  False                  False                  False

   Flood Status_0.472912  Flood Status_0.477228  Flood Status_0.500569  \
0                  False                  False                  False
1                  False                  False                  False
2                  False                  False                  False
3                  False                  False                  False
4                  False                  False                  False

   Flood Status_0.530449 Flood Status_0.566739
0                  False                 False
1                  False                 False
2                  False                 False
3                  False                 False
4                  False                 False

[5 rows x 206 columns]
```

In [7]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data = data[data.columns[:]].apply(le.fit_transform)
```

In [8]:
```
data.head()
```

Out[8]:

	Node	Utilised Bandwith Rate	Packet Drop Rate	Full_Bandwidth	Average_Delay_Time_Per_Sec	Percentage_Of_Lo
0	0	171	24	9	153	
1	1	9	187	0	153	
2	0	195	1	8	70	
3	1	26	171	0	58	
4	0	191	5	7	47	

5 rows × 206 columns

In [9]:
```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = data.drop(columns=['Class'])
y = data['Class']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Further split the training data into training and validation sets (75% train, 25%
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25,

# Print the shapes of the resulting datasets
print("Training set shape:", X_train.shape, y_train.shape)
print("Validation set shape:", X_val.shape, y_val.shape)
print("Test set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (636, 205) (636,)
Validation set shape: (212, 205) (212,)
Test set shape: (212, 205) (212,)
```

In [10]:
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalAveragePooling1D, F
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
from keras.layers import Conv1D, MaxPooling1D, Dropout, GlobalAveragePooling1D, Bat

# Reshape X_train, X_val, and X_test for CNN input
X_train_reshaped = X_train.values.reshape(X_train.shape[0], X_train.shape[1], 1)
X_val_reshaped = X_val.values.reshape(X_val.shape[0], X_val.shape[1], 1)
X_test_reshaped = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)


# Define CNN model with additional convolutional and batch normalization layers
cnn_model = Sequential([
```

```
    Conv1D(filters=64, kernel_size=4, activation='relu', input_shape=(X_train.shape
    MaxPooling1D(pool_size=2),
    Dropout(0.2),  # Add dropout layer

    Conv1D(filters=128, kernel_size=4, activation='relu'),
    MaxPooling1D(pool_size=2),
    Dropout(0.2),  # Add dropout layer
    BatchNormalization(),  # Add batch normalization layer

    Conv1D(filters=256, kernel_size=4, activation='relu'),
    MaxPooling1D(pool_size=2),
    Dropout(0.2),  # Add dropout layer
    BatchNormalization(),  # Add batch normalization layer

    Conv1D(filters=512, kernel_size=4, activation='relu'),
    MaxPooling1D(pool_size=2),
    Dropout(0.2),  # Add dropout layer
    BatchNormalization(),  # Add batch normalization layer

    GlobalAveragePooling1D(),
    Dense(units=4, activation='softmax')  # 4 output classes
])

# Compile the model
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics

# Train the model
history = cnn_model.fit(X_train, y_train, epochs=100, batch_size=32, validation_dat
```

```
C:\Users\A S Computer\AppData\Roaming\Python\Python311\site-packages\keras\src\layer
s\convolutional\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_d
im` argument to a layer. When using Sequential models, prefer using an `Input(shape)
` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 197ms/step - accuracy: 0.4298 - loss: 1.2402 - val_ac
curacy: 0.2170 - val_loss: 1.6830
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 130ms/step - accuracy: 0.5957 - loss: 0.8805 - val_acc
uracy: 0.6085 - val_loss: 1.1156
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 132ms/step - accuracy: 0.6736 - loss: 0.7967 - val_acc
uracy: 0.6934 - val_loss: 0.7554
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 142ms/step - accuracy: 0.7057 - loss: 0.7153 - val_acc
uracy: 0.7406 - val_loss: 0.6401
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 136ms/step - accuracy: 0.7280 - loss: 0.6423 - val_acc
uracy: 0.7453 - val_loss: 0.5849
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 152ms/step - accuracy: 0.7469 - loss: 0.6182 - val_acc
uracy: 0.7925 - val_loss: 0.6038
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 135ms/step - accuracy: 0.7752 - loss: 0.5637 - val_acc
uracy: 0.7264 - val_loss: 0.6795
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 135ms/step - accuracy: 0.7596 - loss: 0.5473 - val_acc
uracy: 0.7311 - val_loss: 0.7372
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 129ms/step - accuracy: 0.8278 - loss: 0.4826 - val_acc
uracy: 0.7170 - val_loss: 0.6913
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 136ms/step - accuracy: 0.8384 - loss: 0.4482 - val_acc
uracy: 0.6415 - val_loss: 0.8655
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 129ms/step - accuracy: 0.8189 - loss: 0.4507 - val_acc
uracy: 0.6038 - val_loss: 0.9553
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 134ms/step - accuracy: 0.8155 - loss: 0.4077 - val_acc
uracy: 0.7736 - val_loss: 0.6052
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 133ms/step - accuracy: 0.8292 - loss: 0.3809 - val_acc
uracy: 0.8302 - val_loss: 0.4951
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 133ms/step - accuracy: 0.8559 - loss: 0.3654 - val_acc
uracy: 0.8302 - val_loss: 0.5094
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 136ms/step - accuracy: 0.8462 - loss: 0.3752 - val_acc
uracy: 0.5802 - val_loss: 0.9473
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 129ms/step - accuracy: 0.8639 - loss: 0.3356 - val_acc
uracy: 0.8491 - val_loss: 0.4706
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 134ms/step - accuracy: 0.8430 - loss: 0.3624 - val_acc
uracy: 0.7028 - val_loss: 0.6964
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 131ms/step - accuracy: 0.8426 - loss: 0.3632 - val_acc
uracy: 0.6792 - val_loss: 0.7859
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 137ms/step - accuracy: 0.8547 - loss: 0.3415 - val_acc
```

```
uracy: 0.6368 - val_loss: 0.8396
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 134ms/step - accuracy: 0.8831 - loss: 0.3198 - val_acc
uracy: 0.6981 - val_loss: 0.7208
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 140ms/step - accuracy: 0.8871 - loss: 0.2762 - val_acc
uracy: 0.7925 - val_loss: 0.5594
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 122ms/step - accuracy: 0.8891 - loss: 0.3697 - val_acc
uracy: 0.8679 - val_loss: 0.4365
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 135ms/step - accuracy: 0.8739 - loss: 0.2968 - val_acc
uracy: 0.8349 - val_loss: 0.3770
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 137ms/step - accuracy: 0.8666 - loss: 0.3375 - val_acc
uracy: 0.8868 - val_loss: 0.3473
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 150ms/step - accuracy: 0.8896 - loss: 0.2914 - val_acc
uracy: 0.8443 - val_loss: 0.3634
Epoch 26/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 143ms/step - accuracy: 0.8902 - loss: 0.2737 - val_acc
uracy: 0.8821 - val_loss: 0.3637
Epoch 27/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 133ms/step - accuracy: 0.8978 - loss: 0.2626 - val_acc
uracy: 0.8868 - val_loss: 0.3094
Epoch 28/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 133ms/step - accuracy: 0.8790 - loss: 0.2925 - val_acc
uracy: 0.8915 - val_loss: 0.2946
Epoch 29/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 6s 161ms/step - accuracy: 0.9013 - loss: 0.2672 - val_acc
uracy: 0.8915 - val_loss: 0.3040
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 138ms/step - accuracy: 0.8863 - loss: 0.2866 - val_acc
uracy: 0.8962 - val_loss: 0.2601
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 131ms/step - accuracy: 0.9029 - loss: 0.2203 - val_acc
uracy: 0.8349 - val_loss: 0.3473
Epoch 32/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 129ms/step - accuracy: 0.9015 - loss: 0.2409 - val_acc
uracy: 0.8821 - val_loss: 0.3322
Epoch 33/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 141ms/step - accuracy: 0.8922 - loss: 0.2632 - val_acc
uracy: 0.8962 - val_loss: 0.2813
Epoch 34/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 125ms/step - accuracy: 0.9022 - loss: 0.2479 - val_acc
uracy: 0.8302 - val_loss: 0.3911
Epoch 35/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.9211 - loss: 0.2225 - val_acc
uracy: 0.8396 - val_loss: 0.3368
Epoch 36/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 133ms/step - accuracy: 0.9005 - loss: 0.2576 - val_acc
uracy: 0.9057 - val_loss: 0.2256
Epoch 37/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 132ms/step - accuracy: 0.9150 - loss: 0.2215 - val_acc
uracy: 0.9009 - val_loss: 0.2337
Epoch 38/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 119ms/step - accuracy: 0.9061 - loss: 0.2316 - val_acc
uracy: 0.9198 - val_loss: 0.2291
Epoch 39/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 126ms/step - accuracy: 0.9240 - loss: 0.2198 - val_acc
uracy: 0.9245 - val_loss: 0.2227
Epoch 40/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 127ms/step - accuracy: 0.8918 - loss: 0.2407 - val_acc
uracy: 0.9009 - val_loss: 0.2585
Epoch 41/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 133ms/step - accuracy: 0.9053 - loss: 0.2096 - val_acc
uracy: 0.8868 - val_loss: 0.2704
Epoch 42/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 145ms/step - accuracy: 0.9424 - loss: 0.1803 - val_acc
uracy: 0.9245 - val_loss: 0.2109
Epoch 43/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 134ms/step - accuracy: 0.9082 - loss: 0.2308 - val_acc
uracy: 0.9292 - val_loss: 0.1924
Epoch 44/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 122ms/step - accuracy: 0.8890 - loss: 0.2698 - val_acc
uracy: 0.8821 - val_loss: 0.2301
Epoch 45/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 134ms/step - accuracy: 0.9127 - loss: 0.2063 - val_acc
uracy: 0.9245 - val_loss: 0.2233
Epoch 46/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 150ms/step - accuracy: 0.9185 - loss: 0.2215 - val_acc
uracy: 0.9057 - val_loss: 0.2324
Epoch 47/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 144ms/step - accuracy: 0.9287 - loss: 0.1861 - val_acc
uracy: 0.9009 - val_loss: 0.1949
Epoch 48/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 131ms/step - accuracy: 0.9186 - loss: 0.2243 - val_acc
uracy: 0.9198 - val_loss: 0.2097
Epoch 49/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 126ms/step - accuracy: 0.9154 - loss: 0.2004 - val_acc
uracy: 0.9104 - val_loss: 0.2054
Epoch 50/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 126ms/step - accuracy: 0.9215 - loss: 0.2034 - val_acc
uracy: 0.8679 - val_loss: 0.2391
Epoch 51/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 142ms/step - accuracy: 0.9195 - loss: 0.1790 - val_acc
uracy: 0.9057 - val_loss: 0.2104
Epoch 52/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 139ms/step - accuracy: 0.9065 - loss: 0.2233 - val_acc
uracy: 0.8962 - val_loss: 0.2063
Epoch 53/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 121ms/step - accuracy: 0.9436 - loss: 0.1675 - val_acc
uracy: 0.8679 - val_loss: 0.2359
Epoch 54/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 126ms/step - accuracy: 0.9300 - loss: 0.1524 - val_acc
uracy: 0.9245 - val_loss: 0.1963
Epoch 55/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 125ms/step - accuracy: 0.9216 - loss: 0.1749 - val_acc
uracy: 0.9198 - val_loss: 0.1809
Epoch 56/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 141ms/step - accuracy: 0.9226 - loss: 0.1703 - val_acc
uracy: 0.9434 - val_loss: 0.1473
```

```
Epoch 57/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 141ms/step - accuracy: 0.9170 - loss: 0.1910 - val_acc
uracy: 0.9292 - val_loss: 0.1814
Epoch 58/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 124ms/step - accuracy: 0.9430 - loss: 0.1573 - val_acc
uracy: 0.9340 - val_loss: 0.1790
Epoch 59/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 122ms/step - accuracy: 0.9533 - loss: 0.1363 - val_acc
uracy: 0.9387 - val_loss: 0.1530
Epoch 60/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 135ms/step - accuracy: 0.9526 - loss: 0.1423 - val_acc
uracy: 0.9292 - val_loss: 0.1786
Epoch 61/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 136ms/step - accuracy: 0.9305 - loss: 0.1869 - val_acc
uracy: 0.9151 - val_loss: 0.1619
Epoch 62/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 131ms/step - accuracy: 0.9237 - loss: 0.1919 - val_acc
uracy: 0.9104 - val_loss: 0.1574
Epoch 63/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 122ms/step - accuracy: 0.9300 - loss: 0.1626 - val_acc
uracy: 0.9434 - val_loss: 0.1539
Epoch 64/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step - accuracy: 0.9156 - loss: 0.1921 - val_acc
uracy: 0.9198 - val_loss: 0.1934
Epoch 65/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 126ms/step - accuracy: 0.9349 - loss: 0.1420 - val_acc
uracy: 0.9340 - val_loss: 0.1662
Epoch 66/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 132ms/step - accuracy: 0.9220 - loss: 0.1583 - val_acc
uracy: 0.9528 - val_loss: 0.1305
Epoch 67/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 136ms/step - accuracy: 0.9144 - loss: 0.2026 - val_acc
uracy: 0.9481 - val_loss: 0.1275
Epoch 68/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 133ms/step - accuracy: 0.9295 - loss: 0.1612 - val_acc
uracy: 0.9292 - val_loss: 0.1496
Epoch 69/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 128ms/step - accuracy: 0.9370 - loss: 0.1608 - val_acc
uracy: 0.9481 - val_loss: 0.1394
Epoch 70/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 136ms/step - accuracy: 0.9193 - loss: 0.1890 - val_acc
uracy: 0.9245 - val_loss: 0.1551
Epoch 71/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 6s 170ms/step - accuracy: 0.9541 - loss: 0.1264 - val_acc
uracy: 0.9387 - val_loss: 0.1408
Epoch 72/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 143ms/step - accuracy: 0.9267 - loss: 0.1525 - val_acc
uracy: 0.9340 - val_loss: 0.1750
Epoch 73/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 126ms/step - accuracy: 0.9528 - loss: 0.1400 - val_acc
uracy: 0.9434 - val_loss: 0.1194
Epoch 74/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 131ms/step - accuracy: 0.9223 - loss: 0.1551 - val_acc
uracy: 0.9575 - val_loss: 0.1334
Epoch 75/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 138ms/step - accuracy: 0.9356 - loss: 0.1534 - val_acc
```

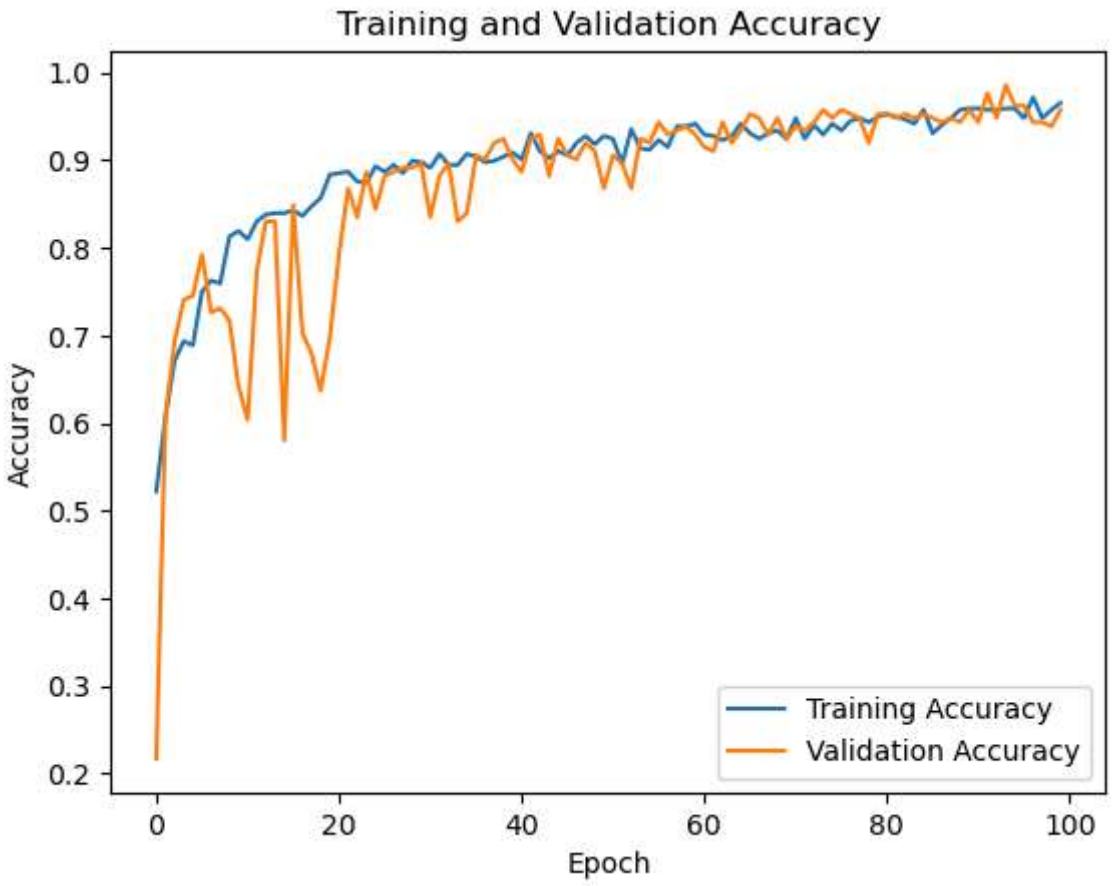
```
uracy: 0.9481 - val_loss: 0.1221
Epoch 76/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 140ms/step - accuracy: 0.9523 - loss: 0.1266 - val_acc
uracy: 0.9575 - val_loss: 0.1195
Epoch 77/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 127ms/step - accuracy: 0.9449 - loss: 0.1357 - val_acc
uracy: 0.9528 - val_loss: 0.1305
Epoch 78/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 132ms/step - accuracy: 0.9504 - loss: 0.1445 - val_acc
uracy: 0.9481 - val_loss: 0.1448
Epoch 79/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 133ms/step - accuracy: 0.9367 - loss: 0.1702 - val_acc
uracy: 0.9198 - val_loss: 0.1335
Epoch 80/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 140ms/step - accuracy: 0.9531 - loss: 0.1116 - val_acc
uracy: 0.9528 - val_loss: 0.1408
Epoch 81/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 136ms/step - accuracy: 0.9537 - loss: 0.1311 - val_acc
uracy: 0.9528 - val_loss: 0.1180
Epoch 82/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 138ms/step - accuracy: 0.9526 - loss: 0.1294 - val_acc
uracy: 0.9481 - val_loss: 0.1205
Epoch 83/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 131ms/step - accuracy: 0.9516 - loss: 0.1416 - val_acc
uracy: 0.9528 - val_loss: 0.1144
Epoch 84/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 145ms/step - accuracy: 0.9587 - loss: 0.1117 - val_acc
uracy: 0.9481 - val_loss: 0.1366
Epoch 85/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 152ms/step - accuracy: 0.9544 - loss: 0.1419 - val_acc
uracy: 0.9528 - val_loss: 0.1311
Epoch 86/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 142ms/step - accuracy: 0.9278 - loss: 0.1530 - val_acc
uracy: 0.9481 - val_loss: 0.1427
Epoch 87/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 134ms/step - accuracy: 0.9452 - loss: 0.1302 - val_acc
uracy: 0.9434 - val_loss: 0.1380
Epoch 88/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 4s 192ms/step - accuracy: 0.9542 - loss: 0.1091 - val_acc
uracy: 0.9481 - val_loss: 0.1445
Epoch 89/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 175ms/step - accuracy: 0.9507 - loss: 0.1292 - val_acc
uracy: 0.9434 - val_loss: 0.1319
Epoch 90/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 137ms/step - accuracy: 0.9525 - loss: 0.1243 - val_acc
uracy: 0.9575 - val_loss: 0.1353
Epoch 91/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 143ms/step - accuracy: 0.9664 - loss: 0.1024 - val_acc
uracy: 0.9434 - val_loss: 0.1277
Epoch 92/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 154ms/step - accuracy: 0.9560 - loss: 0.1194 - val_acc
uracy: 0.9764 - val_loss: 0.1085
Epoch 93/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 158ms/step - accuracy: 0.9607 - loss: 0.1012 - val_acc
uracy: 0.9481 - val_loss: 0.1179
Epoch 94/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 141ms/step - accuracy: 0.9565 - loss: 0.1320 - val_acc
uracy: 0.9858 - val_loss: 0.1151
Epoch 95/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 134ms/step - accuracy: 0.9577 - loss: 0.1033 - val_acc
uracy: 0.9623 - val_loss: 0.1329
Epoch 96/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 135ms/step - accuracy: 0.9531 - loss: 0.1005 - val_acc
uracy: 0.9623 - val_loss: 0.1159
Epoch 97/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 147ms/step - accuracy: 0.9795 - loss: 0.0649 - val_acc
uracy: 0.9434 - val_loss: 0.1411
Epoch 98/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 139ms/step - accuracy: 0.9373 - loss: 0.1440 - val_acc
uracy: 0.9434 - val_loss: 0.1254
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 148ms/step - accuracy: 0.9639 - loss: 0.0865 - val_acc
uracy: 0.9387 - val_loss: 0.1720
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 153ms/step - accuracy: 0.9763 - loss: 0.0780 - val_acc
uracy: 0.9575 - val_loss: 0.1057
```

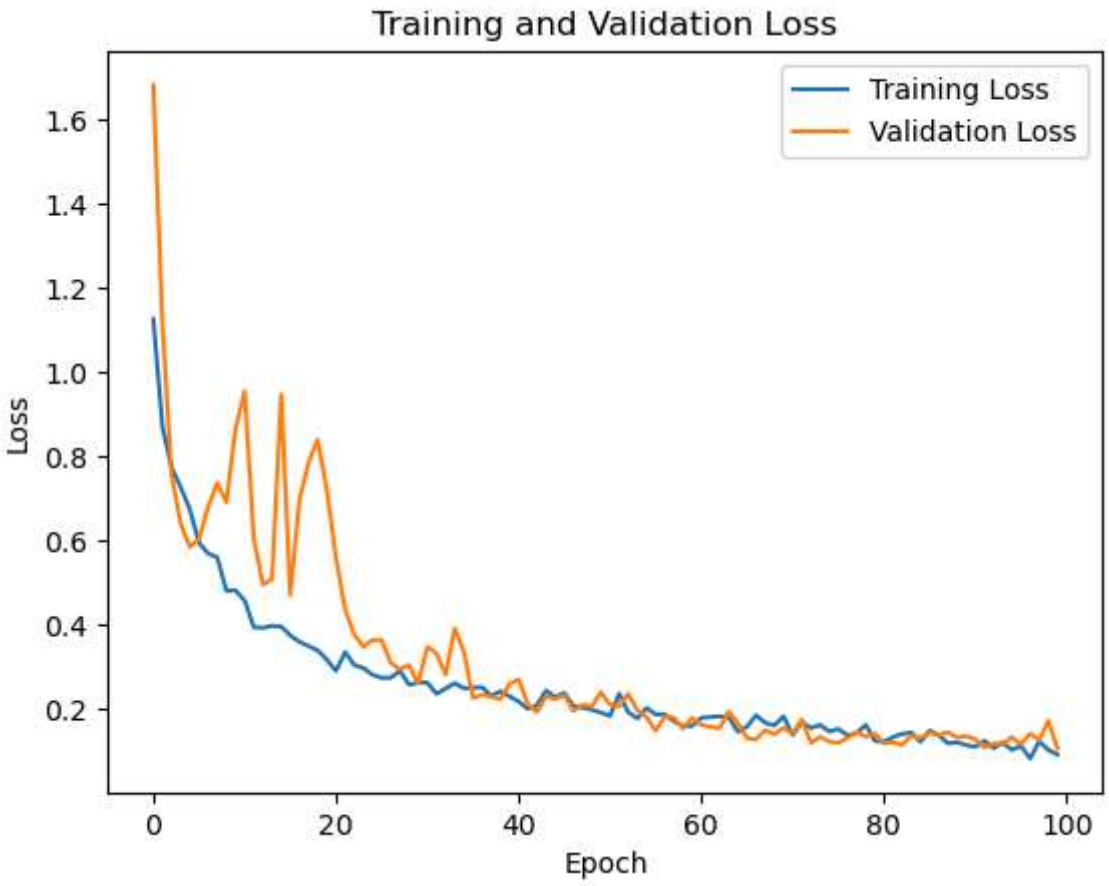
In [11]:

```python
import matplotlib.pyplot as plt

# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [12]:
```
# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [13]:
```
# Generate predictions
predictions = cnn_model.predict(X_test_reshaped)
predicted_classes = np.argmax(predictions, axis=1)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, predicted_classes))
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 155ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        26
           1       1.00      0.90      0.95        99
           2       0.84      0.94      0.89        54
           3       1.00      1.00      1.00        33

    accuracy                           0.94       212
   macro avg       0.93      0.96      0.94       212
weighted avg       0.95      0.94      0.94       212
```

In [14]:
```
# Reshape the training, validation, and test sets for RNN
X_train_rnn = np.reshape(X_train.values, (X_train.shape[0], X_train.shape[1], 1))
X_val_rnn = np.reshape(X_val.values, (X_val.shape[0], X_val.shape[1], 1))
X_test_rnn = np.reshape(X_test.values, (X_test.shape[0], X_test.shape[1], 1))

# Print the shapes of the reshaped datasets
print("Training set shape (RNN):", X_train_rnn.shape, y_train.shape)
```

```
print("Validation set shape (RNN):", X_val_rnn.shape, y_val.shape)
print("Test set shape (RNN):", X_test_rnn.shape, y_test.shape)

from keras.utils import to_categorical

# One-hot encode the target labels
y_train_encoded = to_categorical(y_train)
y_val_encoded = to_categorical(y_val)
y_test_encoded = to_categorical(y_test)
```

```
Training set shape (RNN): (636, 205, 1) (636,)
Validation set shape (RNN): (212, 205, 1) (212,)
Test set shape (RNN): (212, 205, 1) (212,)
```

In [18]:

```
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Dropout
from keras.optimizers import Adam
from keras.layers import LSTM

from keras.optimizers import Adam

# Define the optimizer with the learning rate argument
optimizer = Adam(learning_rate=0.001)

# Define RNN model with additional layers
rnn_model = Sequential([
    SimpleRNN(units=128, activation='relu', input_shape=(X_train_rnn.shape[1], X_tr
    Dropout(0.2),  # Add dropout layer
    SimpleRNN(units=128, activation='relu', return_sequences=True),
    Dropout(0.2),  # Add dropout layer
    SimpleRNN(units=128, activation='relu', return_sequences=True),
    Dropout(0.2),  # Add dropout layer
    SimpleRNN(units=128, activation='relu'),  # Last layer does not need return_seq
    Dropout(0.2),  # Add dropout layer
    Dense(units=64, activation='relu'),  # Additional dense layer
    Dropout(0.2),  # Add dropout layer
    Dense(units=4, activation='softmax')  # 4 output classes
])

# Compile the model with the defined optimizer
rnn_model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['a

# Print model summary
print(rnn_model.summary())
```

```
Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
simple_rnn_12 (SimpleRNN)	(None, 205, 128)	16,640
dropout_19 (Dropout)	(None, 205, 128)	0
simple_rnn_13 (SimpleRNN)	(None, 205, 128)	32,896
dropout_20 (Dropout)	(None, 205, 128)	0
simple_rnn_14 (SimpleRNN)	(None, 205, 128)	32,896
dropout_21 (Dropout)	(None, 205, 128)	0
simple_rnn_15 (SimpleRNN)	(None, 128)	32,896
dropout_22 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 64)	8,256
dropout_23 (Dropout)	(None, 64)	0
dense_8 (Dense)	(None, 4)	260

Total params: 123,844 (483.77 KB)

Trainable params: 123,844 (483.77 KB)

Non-trainable params: 0 (0.00 B)

None

In [20]:
```
from keras.optimizers import Adam

# Define the optimizer with the learning_rate argument
optimizer = Adam(learning_rate=0.001)

# Compile the model with the defined optimizer
rnn_model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['a

# Then, train the model with the encoded target labels
history = rnn_model.fit(X_train_rnn, y_train_encoded,
                        epochs=100,
                        batch_size=32,
                        validation_data=(X_val_rnn, y_val_encoded))
```

```
Epoch 1/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 24s 449ms/step - accuracy: 0.4062 - loss: 1.3342 - val_ac
curacy: 0.5236 - val_loss: 1.1801
Epoch 2/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 368ms/step - accuracy: 0.5420 - loss: 1.1038 - val_acc
uracy: 0.5660 - val_loss: 1.0010
Epoch 3/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 386ms/step - accuracy: 0.5082 - loss: 1.0982 - val_acc
uracy: 0.5283 - val_loss: 1.0239
Epoch 4/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 9s 436ms/step - accuracy: 0.5626 - loss: 0.9894 - val_acc
uracy: 0.5708 - val_loss: 0.9397
Epoch 5/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 11s 454ms/step - accuracy: 0.6006 - loss: 0.9146 - val_ac
curacy: 0.5708 - val_loss: 0.9496
Epoch 6/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 499ms/step - accuracy: 0.5816 - loss: 0.9514 - val_ac
curacy: 0.5849 - val_loss: 0.9103
Epoch 7/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 460ms/step - accuracy: 0.5359 - loss: 0.9474 - val_ac
curacy: 0.5943 - val_loss: 0.9319
Epoch 8/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 428ms/step - accuracy: 0.5999 - loss: 0.9092 - val_ac
curacy: 0.5896 - val_loss: 0.9007
Epoch 9/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 9s 471ms/step - accuracy: 0.5401 - loss: 0.9340 - val_acc
uracy: 0.5802 - val_loss: 0.9051
Epoch 10/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 9s 431ms/step - accuracy: 0.5577 - loss: 0.9492 - val_acc
uracy: 0.5849 - val_loss: 0.9025
Epoch 11/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 598ms/step - accuracy: 0.5506 - loss: 0.9389 - val_ac
curacy: 0.5849 - val_loss: 0.9035
Epoch 12/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 19s 474ms/step - accuracy: 0.5899 - loss: 0.8936 - val_ac
curacy: 0.5896 - val_loss: 0.8939
Epoch 13/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 626ms/step - accuracy: 0.6082 - loss: 0.8717 - val_ac
curacy: 0.5660 - val_loss: 0.9248
Epoch 14/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 19s 516ms/step - accuracy: 0.5608 - loss: 0.9375 - val_ac
curacy: 0.5708 - val_loss: 0.9070
Epoch 15/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 9s 437ms/step - accuracy: 0.5593 - loss: 0.9315 - val_acc
uracy: 0.5896 - val_loss: 0.8914
Epoch 16/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 14s 613ms/step - accuracy: 0.5281 - loss: 0.9377 - val_ac
curacy: 0.5896 - val_loss: 0.8909
Epoch 17/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 18s 463ms/step - accuracy: 0.6061 - loss: 0.8729 - val_ac
curacy: 0.5896 - val_loss: 0.9028
Epoch 18/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 11s 493ms/step - accuracy: 0.6067 - loss: 0.9066 - val_ac
curacy: 0.5849 - val_loss: 0.9145
Epoch 19/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 11s 507ms/step - accuracy: 0.5656 - loss: 0.9185 - val_ac
```

```
curacy: 0.5849 - val_loss: 0.8947
Epoch 20/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 375ms/step - accuracy: 0.5592 - loss: 0.9138 - val_acc
uracy: 0.5943 - val_loss: 0.8921
Epoch 21/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 635ms/step - accuracy: 0.5618 - loss: 0.9270 - val_ac
curacy: 0.5849 - val_loss: 0.8865
Epoch 22/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 623ms/step - accuracy: 0.5802 - loss: 0.8977 - val_ac
curacy: 0.5943 - val_loss: 0.8973
Epoch 23/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 592ms/step - accuracy: 0.5882 - loss: 0.8888 - val_ac
curacy: 0.5802 - val_loss: 0.8896
Epoch 24/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 17s 387ms/step - accuracy: 0.5829 - loss: 0.9002 - val_ac
curacy: 0.5943 - val_loss: 0.8877
Epoch 25/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 371ms/step - accuracy: 0.5621 - loss: 0.9210 - val_acc
uracy: 0.5943 - val_loss: 0.8824
Epoch 26/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 399ms/step - accuracy: 0.5506 - loss: 0.8924 - val_acc
uracy: 0.5660 - val_loss: 0.9176
Epoch 27/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 395ms/step - accuracy: 0.5840 - loss: 0.8847 - val_acc
uracy: 0.5755 - val_loss: 0.9003
Epoch 28/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 386ms/step - accuracy: 0.5562 - loss: 0.9236 - val_acc
uracy: 0.5943 - val_loss: 0.8866
Epoch 29/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 365ms/step - accuracy: 0.5779 - loss: 0.9065 - val_ac
curacy: 0.5802 - val_loss: 0.8894
Epoch 30/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 361ms/step - accuracy: 0.5864 - loss: 0.8786 - val_acc
uracy: 0.5896 - val_loss: 0.8938
Epoch 31/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 13s 510ms/step - accuracy: 0.5943 - loss: 0.8838 - val_ac
curacy: 0.5849 - val_loss: 0.8823
Epoch 32/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 501ms/step - accuracy: 0.5875 - loss: 0.8777 - val_ac
curacy: 0.5896 - val_loss: 0.8865
Epoch 33/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 517ms/step - accuracy: 0.5822 - loss: 0.9028 - val_ac
curacy: 0.5943 - val_loss: 0.8824
Epoch 34/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 383ms/step - accuracy: 0.5737 - loss: 0.8816 - val_acc
uracy: 0.5896 - val_loss: 0.8879
Epoch 35/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 362ms/step - accuracy: 0.5800 - loss: 0.8870 - val_acc
uracy: 0.5991 - val_loss: 0.8541
Epoch 36/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 363ms/step - accuracy: 0.5687 - loss: 0.8983 - val_ac
curacy: 0.5849 - val_loss: 0.8831
Epoch 37/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 374ms/step - accuracy: 0.5862 - loss: 0.8818 - val_acc
uracy: 0.5943 - val_loss: 0.8842
Epoch 38/100
```

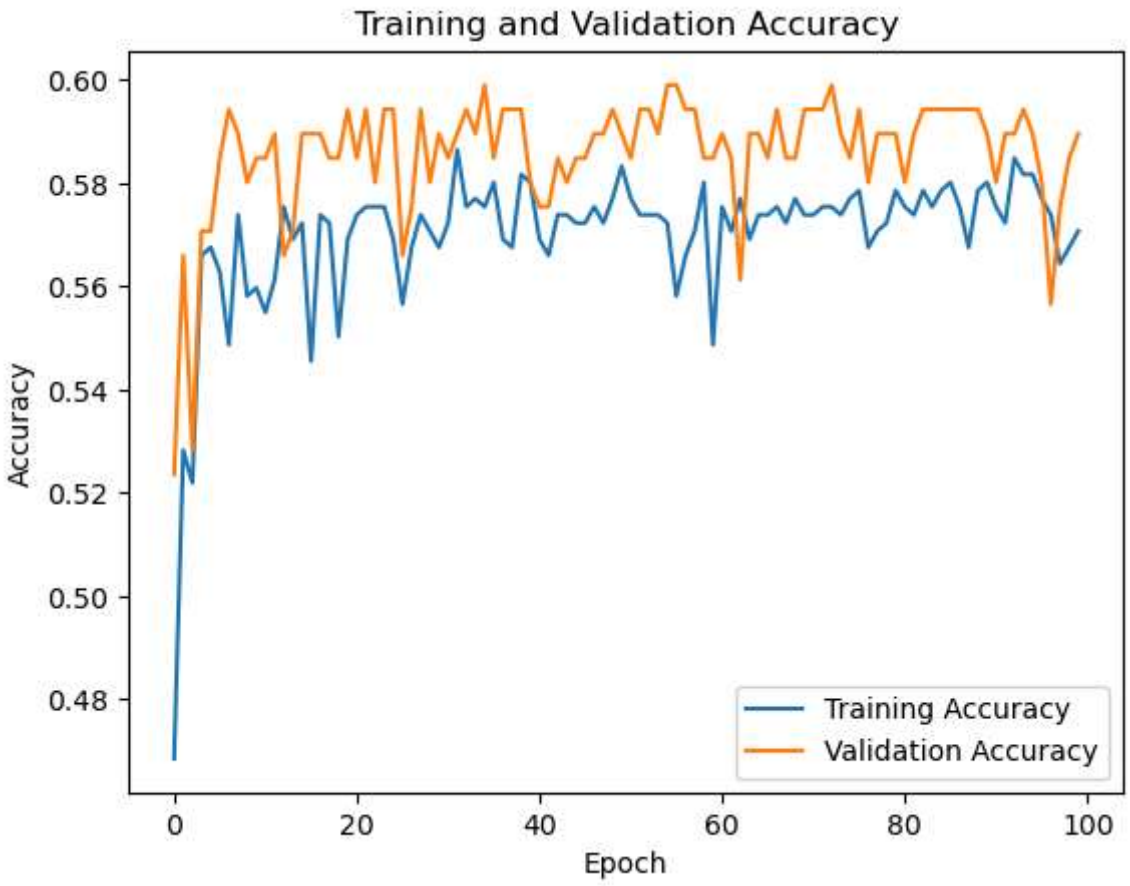
```
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 360ms/step - accuracy: 0.5499 - loss: 0.9118 - val_ac
curacy: 0.5943 - val_loss: 0.8854
Epoch 39/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 354ms/step - accuracy: 0.5707 - loss: 0.8906 - val_acc
uracy: 0.5943 - val_loss: 0.8696
Epoch 40/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 355ms/step - accuracy: 0.5474 - loss: 0.9295 - val_acc
uracy: 0.5802 - val_loss: 0.8927
Epoch 41/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 341ms/step - accuracy: 0.5777 - loss: 0.8979 - val_ac
curacy: 0.5755 - val_loss: 0.9288
Epoch 42/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 371ms/step - accuracy: 0.5418 - loss: 0.9466 - val_acc
uracy: 0.5755 - val_loss: 0.8958
Epoch 43/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 347ms/step - accuracy: 0.5545 - loss: 0.9192 - val_ac
curacy: 0.5849 - val_loss: 0.8917
Epoch 44/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 344ms/step - accuracy: 0.5975 - loss: 0.8686 - val_ac
curacy: 0.5802 - val_loss: 0.8962
Epoch 45/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 11s 363ms/step - accuracy: 0.5666 - loss: 0.9204 - val_ac
curacy: 0.5849 - val_loss: 0.8894
Epoch 46/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 377ms/step - accuracy: 0.5823 - loss: 0.9009 - val_acc
uracy: 0.5849 - val_loss: 0.8888
Epoch 47/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 15s 564ms/step - accuracy: 0.5948 - loss: 0.8805 - val_ac
curacy: 0.5896 - val_loss: 0.8905
Epoch 48/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 394ms/step - accuracy: 0.5771 - loss: 0.8762 - val_acc
uracy: 0.5896 - val_loss: 0.8799
Epoch 49/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 376ms/step - accuracy: 0.5666 - loss: 0.9206 - val_acc
uracy: 0.5943 - val_loss: 0.8783
Epoch 50/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 368ms/step - accuracy: 0.6150 - loss: 0.8701 - val_acc
uracy: 0.5896 - val_loss: 0.8771
Epoch 51/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 372ms/step - accuracy: 0.5750 - loss: 0.8879 - val_acc
uracy: 0.5849 - val_loss: 0.8880
Epoch 52/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 376ms/step - accuracy: 0.5713 - loss: 0.9033 - val_ac
curacy: 0.5943 - val_loss: 0.8821
Epoch 53/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 345ms/step - accuracy: 0.5851 - loss: 0.8788 - val_ac
curacy: 0.5943 - val_loss: 0.8787
Epoch 54/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 360ms/step - accuracy: 0.6089 - loss: 0.8693 - val_acc
uracy: 0.5896 - val_loss: 0.8875
Epoch 55/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 333ms/step - accuracy: 0.5764 - loss: 0.8828 - val_ac
curacy: 0.5991 - val_loss: 0.8712
Epoch 56/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 359ms/step - accuracy: 0.5864 - loss: 0.8908 - val_acc
uracy: 0.5991 - val_loss: 0.9119
```

```
Epoch 57/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 338ms/step - accuracy: 0.5552 - loss: 0.8959 - val_ac
curacy: 0.5943 - val_loss: 0.8826
Epoch 58/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 356ms/step - accuracy: 0.5654 - loss: 0.8855 - val_acc
uracy: 0.5943 - val_loss: 0.8795
Epoch 59/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 350ms/step - accuracy: 0.5868 - loss: 0.8850 - val_acc
uracy: 0.5849 - val_loss: 0.9119
Epoch 60/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 357ms/step - accuracy: 0.5633 - loss: 0.9114 - val_acc
uracy: 0.5849 - val_loss: 0.8802
Epoch 61/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 409ms/step - accuracy: 0.5597 - loss: 0.9031 - val_acc
uracy: 0.5896 - val_loss: 0.8856
Epoch 62/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 414ms/step - accuracy: 0.5659 - loss: 0.8992 - val_acc
uracy: 0.5849 - val_loss: 0.8932
Epoch 63/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 370ms/step - accuracy: 0.5858 - loss: 0.8995 - val_acc
uracy: 0.5613 - val_loss: 0.9304
Epoch 64/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 385ms/step - accuracy: 0.5651 - loss: 0.9113 - val_acc
uracy: 0.5896 - val_loss: 0.8853
Epoch 65/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 343ms/step - accuracy: 0.5896 - loss: 0.8764 - val_ac
curacy: 0.5896 - val_loss: 0.8907
Epoch 66/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 361ms/step - accuracy: 0.5663 - loss: 0.9052 - val_acc
uracy: 0.5849 - val_loss: 0.8836
Epoch 67/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 363ms/step - accuracy: 0.5777 - loss: 0.8986 - val_acc
uracy: 0.5943 - val_loss: 0.8750
Epoch 68/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 358ms/step - accuracy: 0.5566 - loss: 0.9050 - val_acc
uracy: 0.5849 - val_loss: 0.8850
Epoch 69/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 340ms/step - accuracy: 0.5935 - loss: 0.8805 - val_ac
curacy: 0.5849 - val_loss: 0.8871
Epoch 70/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 361ms/step - accuracy: 0.5830 - loss: 0.8662 - val_acc
uracy: 0.5943 - val_loss: 0.8804
Epoch 71/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 367ms/step - accuracy: 0.5847 - loss: 0.8786 - val_acc
uracy: 0.5943 - val_loss: 0.8699
Epoch 72/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 365ms/step - accuracy: 0.5775 - loss: 0.8799 - val_ac
curacy: 0.5943 - val_loss: 0.8711
Epoch 73/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 390ms/step - accuracy: 0.5720 - loss: 0.8699 - val_acc
uracy: 0.5991 - val_loss: 0.9365
Epoch 74/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 365ms/step - accuracy: 0.5655 - loss: 0.9206 - val_ac
curacy: 0.5896 - val_loss: 0.8827
Epoch 75/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 357ms/step - accuracy: 0.5776 - loss: 0.8852 - val_acc
```

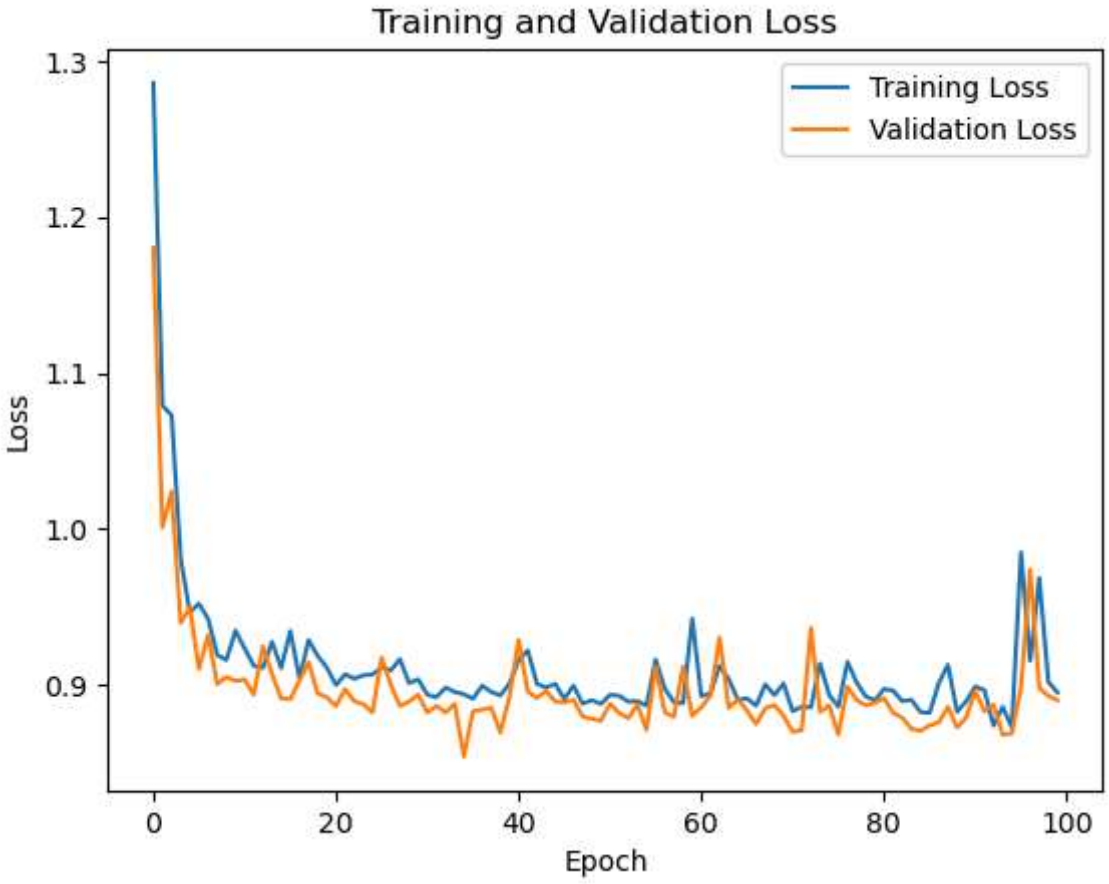
```
uracy: 0.5849 - val_loss: 0.8869
Epoch 76/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 358ms/step - accuracy: 0.5664 - loss: 0.8988 - val_ac
curacy: 0.5943 - val_loss: 0.8683
Epoch 77/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 368ms/step - accuracy: 0.5638 - loss: 0.9111 - val_ac
curacy: 0.5802 - val_loss: 0.8989
Epoch 78/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 339ms/step - accuracy: 0.5744 - loss: 0.8886 - val_ac
curacy: 0.5896 - val_loss: 0.8904
Epoch 79/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 368ms/step - accuracy: 0.5688 - loss: 0.9041 - val_acc
uracy: 0.5896 - val_loss: 0.8869
Epoch 80/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 342ms/step - accuracy: 0.5718 - loss: 0.9007 - val_ac
curacy: 0.5896 - val_loss: 0.8885
Epoch 81/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 376ms/step - accuracy: 0.5878 - loss: 0.8758 - val_acc
uracy: 0.5802 - val_loss: 0.8915
Epoch 82/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 347ms/step - accuracy: 0.5632 - loss: 0.9119 - val_ac
curacy: 0.5896 - val_loss: 0.8821
Epoch 83/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 345ms/step - accuracy: 0.6187 - loss: 0.8459 - val_ac
curacy: 0.5943 - val_loss: 0.8790
Epoch 84/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 361ms/step - accuracy: 0.5881 - loss: 0.8831 - val_acc
uracy: 0.5943 - val_loss: 0.8719
Epoch 85/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 358ms/step - accuracy: 0.5783 - loss: 0.8651 - val_ac
curacy: 0.5943 - val_loss: 0.8706
Epoch 86/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 364ms/step - accuracy: 0.5584 - loss: 0.8823 - val_acc
uracy: 0.5943 - val_loss: 0.8741
Epoch 87/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 342ms/step - accuracy: 0.5751 - loss: 0.8967 - val_ac
curacy: 0.5943 - val_loss: 0.8759
Epoch 88/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 9s 429ms/step - accuracy: 0.5800 - loss: 0.8801 - val_acc
uracy: 0.5943 - val_loss: 0.8860
Epoch 89/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 374ms/step - accuracy: 0.5977 - loss: 0.8670 - val_acc
uracy: 0.5943 - val_loss: 0.8728
Epoch 90/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 11s 386ms/step - accuracy: 0.5669 - loss: 0.8959 - val_ac
curacy: 0.5896 - val_loss: 0.8791
Epoch 91/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 365ms/step - accuracy: 0.5771 - loss: 0.8815 - val_acc
uracy: 0.5802 - val_loss: 0.8961
Epoch 92/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 354ms/step - accuracy: 0.5814 - loss: 0.8821 - val_ac
curacy: 0.5896 - val_loss: 0.8831
Epoch 93/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 351ms/step - accuracy: 0.6075 - loss: 0.8728 - val_acc
uracy: 0.5896 - val_loss: 0.8873
Epoch 94/100
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 342ms/step - accuracy: 0.5878 - loss: 0.8954 - val_ac
curacy: 0.5943 - val_loss: 0.8681
Epoch 95/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 11s 373ms/step - accuracy: 0.6000 - loss: 0.8502 - val_ac
curacy: 0.5896 - val_loss: 0.8689
Epoch 96/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 372ms/step - accuracy: 0.5914 - loss: 0.9114 - val_acc
uracy: 0.5802 - val_loss: 0.8981
Epoch 97/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 13s 644ms/step - accuracy: 0.5789 - loss: 0.9058 - val_ac
curacy: 0.5566 - val_loss: 0.9739
Epoch 98/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 15s 356ms/step - accuracy: 0.5517 - loss: 1.0100 - val_ac
curacy: 0.5755 - val_loss: 0.8982
Epoch 99/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 11s 376ms/step - accuracy: 0.5808 - loss: 0.8936 - val_ac
curacy: 0.5849 - val_loss: 0.8930
Epoch 100/100
20/20 ━━━━━━━━━━━━━━━━━━━━ 10s 346ms/step - accuracy: 0.5730 - loss: 0.8967 - val_ac
curacy: 0.5896 - val_loss: 0.8902
```

In [21]:
```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.show()
```



In [22]:
```
# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



In [23]:
```
# Generate predictions
y_pred = rnn_model.predict(X_test_rnn)
y_pred_classes = np.argmax(y_pred, axis=1)

# Print classification report
print(classification_report(y_test, y_pred_classes))
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 4s 368ms/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       0.55      1.00      0.71        99
           2       1.00      0.11      0.20        54
           3       0.00      0.00      0.00        33

    accuracy                           0.62       212
   macro avg       0.64      0.53      0.48       212
weighted avg       0.63      0.62      0.50       212
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 i
n labels with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [24]:

```python
import pickle

# Save the model in .h5 format
cnn_model.save('cnn_model.h5')

# Save the history (optional)
with open('cnn_model_history.pkl', 'wb') as file:
    pickle.dump(history.history, file)
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.
saving.save_model(model)`. This file format is considered legacy. We recommend using
instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.savin
g.save_model(model, 'my_model.keras')`.
```