In [63]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

In [64]:
```
# Load the ECG data from the text file
file_path = r'C:\Users\diyu2\OneDrive - AUT University\AUT YEAR 4\INDUSTRIAL PRO
```

In [65]:
```
# Reading the file and converting the data into an array
with open(file_path, 'r') as file:
    lines = file.readlines()
```

In [66]:
```
# Process only numeric lines
data = []
for line in lines:
    try:
        # Split the line into numbers and convert to integers
        numbers = list(map(int, line.strip().split()))
        data.append(numbers)
    except ValueError:
        # Ignore lines that can't be converted to integers
        continue

# Determine the maximum length of any line
max_length = max([len(row) for row in data])

# Pad all rows with zeros to ensure they have the same length
padded_data = np.array([np.pad(row, (0, max_length - len(row)), 'constant') for
```

In [67]:
```
# Check the shape and adjust it to fit the CNN input
padded_data = padded_data.reshape((padded_data.shape[0], padded_data.shape[1], 1
```

In [68]:
```
# Generate labels (for simplicity, using random labels for now)
labels = np.random.randint(2, size=padded_data.shape[0])
```

In [69]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_data, labels, test_si

# Build the CNN model
model = Sequential()
```

In [70]:
```
# 1D CNN layer for processing sequential ECG data
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(padd
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())

# Fully connected layers
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Assuming binary classification

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']
```

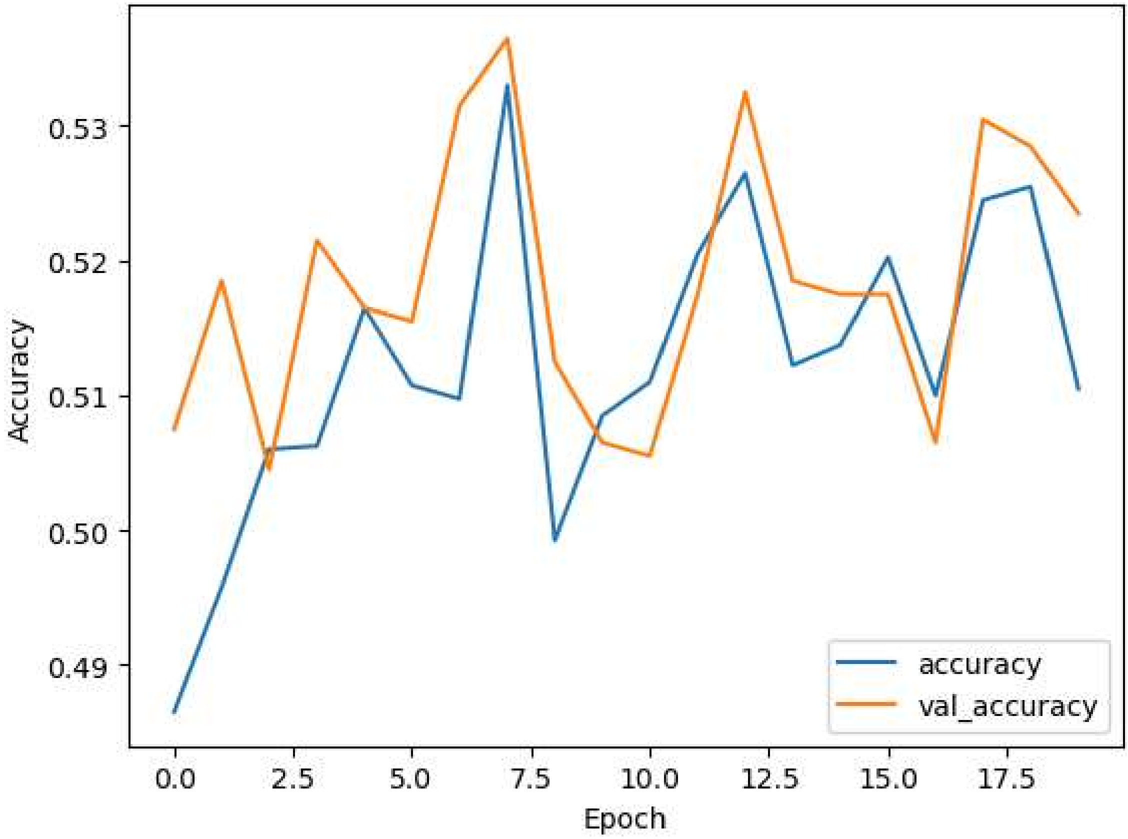
In [71]:
```
# Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test
```

```
Epoch 1/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - accuracy: 0.4885 - loss: 1.6451 - val
_accuracy: 0.5075 - val_loss: 0.8990
Epoch 2/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5032 - loss: 1.0265 - val_
accuracy: 0.5185 - val_loss: 0.7877
Epoch 3/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5029 - loss: 0.8089 - val_
accuracy: 0.5045 - val_loss: 0.8009
Epoch 4/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5054 - loss: 0.7873 - val_
accuracy: 0.5215 - val_loss: 0.6989
Epoch 5/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5148 - loss: 0.7336 - val_
accuracy: 0.5165 - val_loss: 0.6912
Epoch 6/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5167 - loss: 0.7103 - val_
accuracy: 0.5155 - val_loss: 0.6976
Epoch 7/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5159 - loss: 0.7111 - val_
accuracy: 0.5315 - val_loss: 0.7201
Epoch 8/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5382 - loss: 0.7039 - val_
accuracy: 0.5365 - val_loss: 0.7112
Epoch 9/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.4991 - loss: 0.7228 - val_
accuracy: 0.5125 - val_loss: 0.7170
Epoch 10/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5134 - loss: 0.7295 - val_
accuracy: 0.5065 - val_loss: 0.7103
Epoch 11/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5215 - loss: 0.7002 - val_
accuracy: 0.5055 - val_loss: 0.6958
Epoch 12/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5200 - loss: 0.6951 - val_
accuracy: 0.5175 - val_loss: 0.6934
Epoch 13/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5271 - loss: 0.6997 - val_
accuracy: 0.5325 - val_loss: 0.6944
Epoch 14/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5212 - loss: 0.6994 - val_
accuracy: 0.5185 - val_loss: 0.6960
Epoch 15/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5141 - loss: 0.6975 - val_
accuracy: 0.5175 - val_loss: 0.6902
Epoch 16/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5420 - loss: 0.6937 - val_
accuracy: 0.5175 - val_loss: 0.6952
Epoch 17/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5145 - loss: 0.6920 - val_
accuracy: 0.5065 - val_loss: 0.6925
Epoch 18/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.5279 - loss: 0.6961 - val_
accuracy: 0.5305 - val_loss: 0.6907
Epoch 19/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5300 - loss: 0.6914 - val_
accuracy: 0.5285 - val_loss: 0.6864
Epoch 20/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.5175 - loss: 0.6913 - val_
accuracy: 0.5235 - val_loss: 0.6924
```

In [72]:
```
# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_acc}')
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5261 - loss: 0.6910
Test Accuracy: 0.5234765410423279
```

In [73]:
```
# Plotting the training and validation accuracy
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()
```



In [74]:
```
# Save the trained model
model.save('ecg_cnn_model.h5')
```

```
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.saving.save_model(model, 'my_model.keras')`.
```

In [ ]: