In [1]:
```
import os
import numpy as np
from scipy.io import loadmat
from scipy.signal import spectrogram
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dens
import matplotlib.pyplot as plt
```

In [2]:
```
# Function to load ECG data from the folders and convert to spectrogram
def load_ecg_data_from_folders(root_folder, nperseg=256, noverlap=128):
    all_data = []
    all_labels = []

    # Traverse through the root folder and subfolders
    for subdir, dirs, files in os.walk(root_folder):
        print(f"Checking folder: {subdir}")
        for file in files:
            if file.endswith('.mat'):
                file_path = os.path.join(subdir, file)
                print(f"Loading MATLAB file: {file_path}")

                try:
                    # Load the MATLAB file
                    mat_data = loadmat(file_path)
                    print(f"Keys in MATLAB file: {mat_data.keys()}")

                    # Assuming 'val' contains ECG data
                    if 'val' in mat_data:
                        ecg_data = mat_data['val'].squeeze()  # Squeeze

                        # Convert ECG data to spectrogram
                        f, t, Sxx = spectrogram(ecg_data, nperseg=npers
                        Sxx = np.log(Sxx + 1e-8)  # Apply log transform

                        # Resize spectrogram to a fixed size
                        desired_size = (128, 128)  # Example size, adju
                        Sxx_resized = np.resize(Sxx, desired_size)

                        all_data.append(Sxx_resized)
                        all_labels.append(np.random.randint(2))  # Plac
                    else:
                        print(f"'val' key not found in {file_path}")
                except Exception as e:
                    print(f"Error loading MATLAB file {file_path}: {e}"

    return np.array(all_data), np.array(all_labels)
```

In [3]:
```
# Root folder path where your .mat data is stored
root_folder = r'C:\Users\diyu2\OneDrive - AUT University\AUT YEAR 4\IND
```

In [4]:
```
# Load the ECG data from folders
data, labels = load_ecg_data_from_folders(root_folder)
```

```
or-arrhythmia\WFDBRecords\34\348\JS34254.mat
Keys in MATLAB file: dict_keys(['val'])
Loading MATLAB file: C:\Users\diyu2\OneDrive - AUT University\AUT Y
EAR 4\INDUSTRIAL PROJECT (Mechanical)\Reports Part B\ECG database-f
or-arrhythmia\WFDBRecords\34\348\JS34255.mat
Keys in MATLAB file: dict_keys(['val'])
Loading MATLAB file: C:\Users\diyu2\OneDrive - AUT University\AUT Y
EAR 4\INDUSTRIAL PROJECT (Mechanical)\Reports Part B\ECG database-f
or-arrhythmia\WFDBRecords\34\348\JS34256.mat
Keys in MATLAB file: dict_keys(['val'])
Loading MATLAB file: C:\Users\diyu2\OneDrive - AUT University\AUT Y
EAR 4\INDUSTRIAL PROJECT (Mechanical)\Reports Part B\ECG database-f
or-arrhythmia\WFDBRecords\34\348\JS34257.mat
Keys in MATLAB file: dict_keys(['val'])
Loading MATLAB file: C:\Users\diyu2\OneDrive - AUT University\AUT Y
EAR 4\INDUSTRIAL PROJECT (Mechanical)\Reports Part B\ECG database-f
or-arrhythmia\WFDBRecords\34\348\JS34258.mat
Keys in MATLAB file: dict_keys(['val'])
Loading MATLAB file: C:\Users\diyu2\OneDrive - AUT University\AUT Y
EAR 4\INDUSTRIAL PROJECT (Mechanical)\Reports Part B\ECG database-f
```

In [5]:
```
# Check the shape of the data
print(f"Original shape of data: {data.shape}")
```

```
Original shape of data: (45152, 128, 128)
```

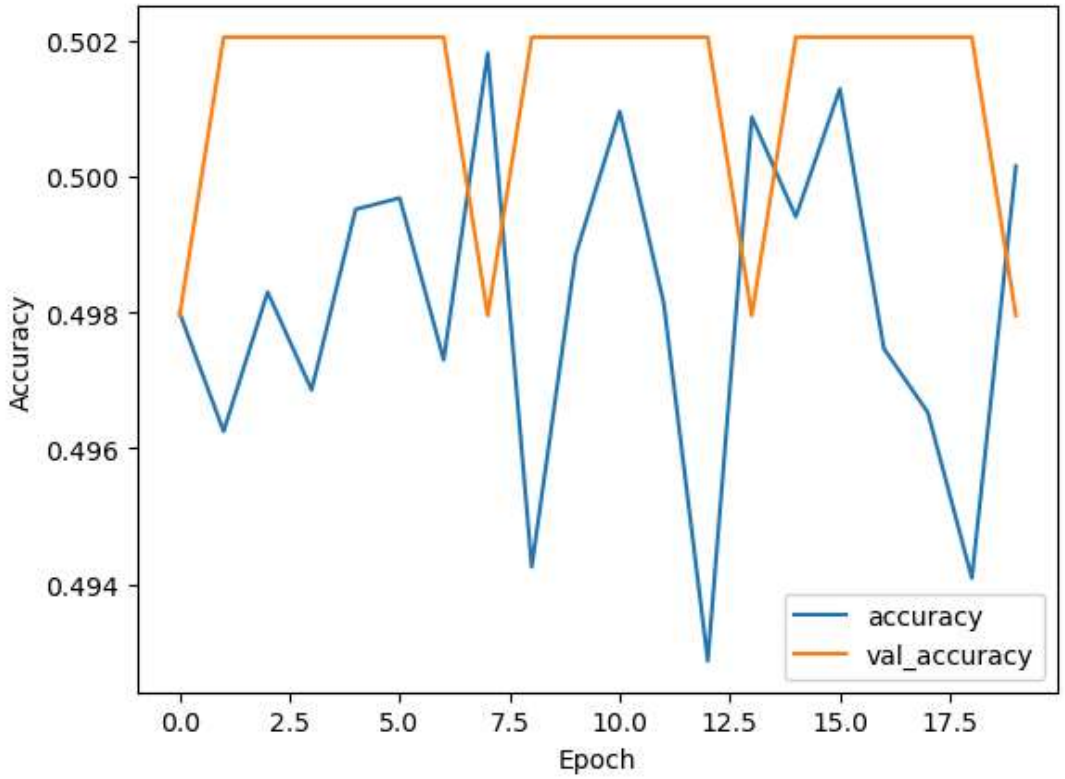
In [6]:

```
# Ensure data is loaded before proceeding
if data.shape[0] > 0:
    # Normalize data to [0, 1]
    data = (data - np.min(data)) / (np.max(data) - np.min(data))

    # Expand dimensions for CNN input (add channel dimension)
    data = np.expand_dims(data, axis=-1)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(data, labels, t

    # Build the CNN model
    model = Sequential()

    # 2D CNN layers for processing spectrogram images
    model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu',
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu')
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'
    model.add(MaxPooling2D(pool_size=(2, 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 classi

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

    # Train the model
    history = model.fit(X_train, y_train, epochs=20, validation_data=(X

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

    # 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()

    # Save the trained model
    model.save('ecg_cnn_model.h5')
else:
    print("No data was loaded, exiting.")
```

```
C:\Users\diyu2\anaconda3\Lib\site-packages\keras\src\layers\convolutio
nal\base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input
_dim` 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, **kwarg
s)
```

```
Epoch 1/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 230s 200ms/step - accuracy: 0.4977 - lo
ss: 0.6935 - val_accuracy: 0.4980 - val_loss: 0.6932
Epoch 2/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 253s 224ms/step - accuracy: 0.4989 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 3/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 264s 233ms/step - accuracy: 0.4952 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 4/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 320s 283ms/step - accuracy: 0.4915 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 5/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 271s 240ms/step - accuracy: 0.5013 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 6/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 329s 292ms/step - accuracy: 0.4981 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 7/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 321s 284ms/step - accuracy: 0.4966 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 8/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 264s 234ms/step - accuracy: 0.5061 - lo
ss: 0.6931 - val_accuracy: 0.4980 - val_loss: 0.6932
Epoch 9/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 290s 257ms/step - accuracy: 0.4915 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 10/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 286s 253ms/step - accuracy: 0.5004 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 11/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 285s 252ms/step - accuracy: 0.5038 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 12/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 288s 255ms/step - accuracy: 0.4960 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 13/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 284s 252ms/step - accuracy: 0.4933 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 14/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 274s 242ms/step - accuracy: 0.5054 - lo
ss: 0.6931 - val_accuracy: 0.4980 - val_loss: 0.6932
Epoch 15/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 277s 245ms/step - accuracy: 0.4970 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 16/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 244s 216ms/step - accuracy: 0.5039 - lo
ss: 0.6931 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 17/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 254s 225ms/step - accuracy: 0.4956 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6932
Epoch 18/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 258s 228ms/step - accuracy: 0.5007 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 19/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 245s 217ms/step - accuracy: 0.4951 - lo
ss: 0.6932 - val_accuracy: 0.5020 - val_loss: 0.6931
Epoch 20/20
1129/1129 ━━━━━━━━━━━━━━━━━━━━ 249s 220ms/step - accuracy: 0.5040 - lo
ss: 0.6931 - val_accuracy: 0.4980 - val_loss: 0.6932
283/283 ━━━━━━━━━━━━━━━━━━━━ 18s 64ms/step - accuracy: 0.4968 - loss:
```

```
0.6932
Test Accuracy: 0.4979515075683594
```




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