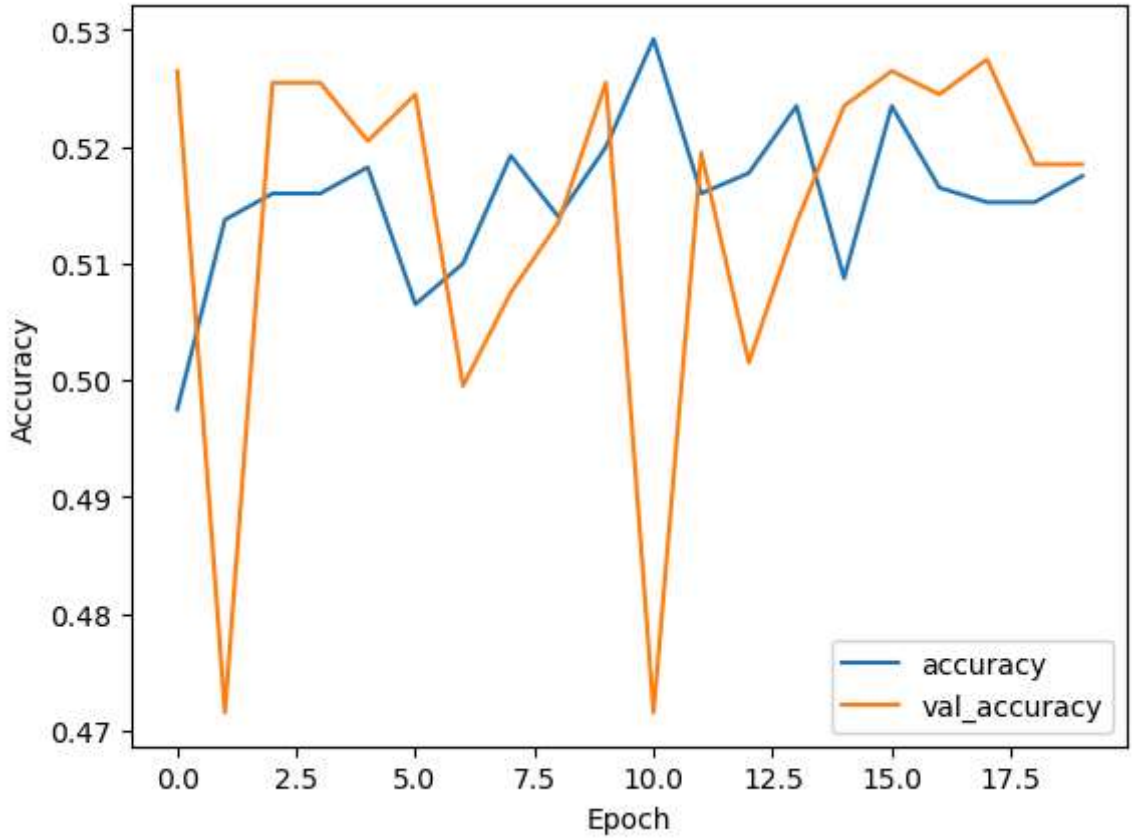
In [3]:
```
import numpy as np
import cv2
from scipy import signal
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

def convert_to_spectrogram(data, fs=1000, nperseg=256):
    """
    Convert ECG time-series data to spectrogram images.

    :param data: Raw ECG data as a list of numpy arrays.
    :param fs: Sampling frequency of the ECG data.
    :param nperseg: Length of each segment for the spectrogram.
    :return: Spectrogram images as a numpy array.
    """
    spectrogram_images = []
    for signal_data in data:
        if len(signal_data) < nperseg:
            # Pad the signal if it's shorter than nperseg
            signal_data = np.pad(signal_data, (0, nperseg - len(signal_data)), '
        f, t, Sxx = signal.spectrogram(signal_data, fs=fs, nperseg=nperseg)
        # Normalize the spectrogram
        Sxx = np.log(Sxx + 1e-8)
        # Resize spectrogram to fixed size
        img = cv2.resize(Sxx, (128, 128))
        spectrogram_images.append(img)
    return np.array(spectrogram_images)

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

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

# 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

# Convert the ECG data to spectrogram images
spectrogram_images = convert_to_spectrogram(data)

# Check the shape of the images and normalize pixel values
spectrogram_images = spectrogram_images.reshape((spectrogram_images.shape[0], 12
spectrogram_images = spectrogram_images.astype('float32') / np.max(spectrogram_i

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

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

# Build the CNN model for image data
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', input_shape=
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
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']

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

# 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')
```

```
C:\Users\diyu2\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_co
nv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a la
yer. When using Sequential models, prefer using an `Input(shape)` object as the f
irst layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 50s 348ms/step - accuracy: 0.4871 - loss: 3.5732 - v
al_accuracy: 0.5265 - val_loss: 0.7074
Epoch 2/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 44s 350ms/step - accuracy: 0.5315 - loss: 0.6950 - v
al_accuracy: 0.4715 - val_loss: 0.6945
Epoch 3/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 41s 327ms/step - accuracy: 0.5191 - loss: 0.6915 - v
al_accuracy: 0.5255 - val_loss: 0.6927
Epoch 4/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 40s 319ms/step - accuracy: 0.5189 - loss: 0.6928 - v
al_accuracy: 0.5255 - val_loss: 0.6931
Epoch 5/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 40s 318ms/step - accuracy: 0.5262 - loss: 0.6918 - v
al_accuracy: 0.5205 - val_loss: 0.6917
Epoch 6/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 43s 340ms/step - accuracy: 0.5157 - loss: 0.6906 - v
al_accuracy: 0.5245 - val_loss: 0.6926
Epoch 7/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 43s 341ms/step - accuracy: 0.5036 - loss: 0.6952 - v
al_accuracy: 0.4995 - val_loss: 0.6934
Epoch 8/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 43s 342ms/step - accuracy: 0.5176 - loss: 0.6919 - v
al_accuracy: 0.5075 - val_loss: 0.6935
Epoch 9/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 43s 341ms/step - accuracy: 0.5276 - loss: 0.6911 - v
al_accuracy: 0.5135 - val_loss: 0.6926
Epoch 10/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 41s 325ms/step - accuracy: 0.5358 - loss: 0.6922 - v
al_accuracy: 0.5255 - val_loss: 0.6931
Epoch 11/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 40s 320ms/step - accuracy: 0.5349 - loss: 0.6903 - v
al_accuracy: 0.4715 - val_loss: 0.6988
Epoch 12/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 40s 314ms/step - accuracy: 0.5241 - loss: 0.6954 - v
al_accuracy: 0.5195 - val_loss: 0.6929
Epoch 13/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 40s 317ms/step - accuracy: 0.5212 - loss: 0.6923 - v
al_accuracy: 0.5015 - val_loss: 0.6933
Epoch 14/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 49s 392ms/step - accuracy: 0.5267 - loss: 0.6941 - v
al_accuracy: 0.5135 - val_loss: 0.6948
Epoch 15/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 52s 411ms/step - accuracy: 0.5163 - loss: 0.6912 - v
al_accuracy: 0.5235 - val_loss: 0.6920
Epoch 16/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 47s 373ms/step - accuracy: 0.5196 - loss: 0.6921 - v
al_accuracy: 0.5265 - val_loss: 0.6934
Epoch 17/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 42s 334ms/step - accuracy: 0.5149 - loss: 0.6927 - v
al_accuracy: 0.5245 - val_loss: 0.6926
Epoch 18/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 41s 323ms/step - accuracy: 0.5198 - loss: 0.6899 - v
al_accuracy: 0.5275 - val_loss: 0.6930
Epoch 19/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 40s 315ms/step - accuracy: 0.5270 - loss: 0.6891 - v
al_accuracy: 0.5185 - val_loss: 0.6920
Epoch 20/20
126/126 ━━━━━━━━━━━━━━━━━━━━ 40s 317ms/step - accuracy: 0.5248 - loss: 0.6892 - v
al_accuracy: 0.5185 - val_loss: 0.6925
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.5315 - loss: 0.6928
Test Accuracy: 0.5184814929962158
```




```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.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 `ke
ras.saving.save_model(model, 'my_model.keras')`.
```

In [ ]: