```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
```

```
import os
import shutil
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import glob
from tqdm import tqdm
import cv2

# Define paths
dataset_path = "/content/drive/MyDrive/samples_CV"  # Folder with all .wav files
output_path = "/content/drive/MyDrive/processed_images"  # Where processed images will be saved
classified_path = "/content/drive/MyDrive/samples_CV_classified"  # Temporary folder to classify files

# Define class names
classes = ["Arcing", "Corona", "Looseness", "Tracking"]

# Create output and classified folders for each class
for cls in classes:
    os.makedirs(os.path.join(output_path, cls), exist_ok=True)
    os.makedirs(os.path.join(classified_path, cls), exist_ok=True)

# Automatically classify .wav files based on filename substring
wav_files = glob.glob(os.path.join(dataset_path, "*.wav"))
for file in wav_files:
    basename = os.path.basename(file)
    assigned = False
    for cls in classes:
        if cls.lower() in basename.lower():
            shutil.copy(file, os.path.join(classified_path, cls, basename))
            assigned = True
            break
    if not assigned:
        print(f"File {basename} did not match any class!")

# Audio processing parameters
sample_rate = 22050       # Sample rate
segment_duration = 0.5    # Segment duration in seconds
overlap = 0.2             # Overlap duration in seconds
hop_length = 512          # Number of samples between successive frames
n_fft = 1024              # FFT window size

def save_spectrogram(audio_path, save_dir, filename_prefix):
    # Load audio using the renamed sample_rate variable
    y, _ = librosa.load(audio_path, sr=sample_rate)
    segment_samples = int(segment_duration * sample_rate)
    overlap_samples = int(overlap * sample_rate)

    start = 0
    count = 0
    while start + segment_samples < len(y):
        segment = y[start:start + segment_samples]
        start += segment_samples - overlap_samples

        # Compute mel-spectrogram and convert to decibel scale
        S = librosa.feature.melspectrogram(y=segment, sr=sample_rate, n_fft=n_fft, hop_length=hop_length)
        S_db = librosa.power_to_db(S, ref=np.max)

        # Plot spectrogram and save as a square image
        fig, ax = plt.subplots(figsize=(3, 3))  # Square figure
        librosa.display.specshow(S_db, sr=sample_rate, hop_length=hop_length, x_axis=None, y_axis=None, ax=ax)
        ax.set_xticks([])
        ax.set_yticks([])
        ax.axis('off')

        img_path = os.path.join(save_dir, f"{filename_prefix}_{count}.png")
        plt.savefig(img_path, bbox_inches='tight', pad_inches=0, dpi=100)
```

```
            plt.close(fig)

            # Ensure the saved image is square (resize if needed)
            img = cv2.imread(img_path)
            if img is not None:
                img_resized = cv2.resize(img, (256, 256))
                cv2.imwrite(img_path, img_resized)
            count += 1

    # Process audio files in each classified folder
    for cls in classes:
        class_folder = os.path.join(classified_path, cls)
        audio_files = glob.glob(os.path.join(class_folder, "*.wav"))
        for audio_file in tqdm(audio_files, desc=f"Processing {cls}"):
            filename_prefix = os.path.splitext(os.path.basename(audio_file))[0]
            save_spectrogram(audio_file, os.path.join(output_path, cls), filename_prefix)

    print("Processing complete! Images saved in", output_path)
```

```
Processing Arcing: 100%|██████████| 4/4 [00:54<00:00, 13.58s/it]
Processing Corona: 100%|██████████| 12/12 [00:48<00:00,  4.07s/it]
Processing Looseness: 100%|██████████| 8/8 [00:56<00:00,  7.05s/it]
Processing Tracking: 100%|██████████| 11/11 [00:54<00:00,  5.00s/it]Processing complete! Images saved in /content/drive/M
```

```
    import os
    import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, InputLayer
    from tensorflow.keras.optimizers import Adam

    # Define paths and parameters
    data_dir = "/content/drive/MyDrive/processed_images"  # Folder with processed images
    img_height, img_width = 256, 256
    batch_size = 16

    # Create ImageDataGenerators for training and validation
    datagen = ImageDataGenerator(
        rescale=1./255,         # Normalize images
        validation_split=0.2,   # 20% for validation
    )

    train_generator = datagen.flow_from_directory(
        data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical',
        subset='training',
        shuffle=True,
    )

    validation_generator = datagen.flow_from_directory(
        data_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation',
        shuffle=False,
    )

    # Dynamically set the number of classes based on the data
    num_classes = len(train_generator.class_indices)
    print(f"Number of classes detected: {num_classes}")

    # Build a simple CNN model using an InputLayer for clarity
    model = Sequential([
        InputLayer(input_shape=(img_height, img_width, 3)),
        Conv2D(32, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),

        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
```

```
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),

        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')  # Adjusted to match detected classes
    ])

    # Compile the model using the Adam optimizer
    optimizer = Adam(learning_rate=1e-4)
    model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

    # Display the model summary
    model.summary()

    # Train the model
    epochs = 20
    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator
    )

    # Save the trained model (optional)
    model.save("/content/drive/MyDrive/trained_model.h5")
```

```
Found 1588 images belonging to 6 classes.
Found 394 images belonging to 6 classes.
Number of classes detected: 6
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/input_layer.py:27: UserWarning: Argument `input_shape` is d
  warnings.warn(
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_3 (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_4 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_5 (Conv2D)	(None, 60, 60, 128)	73,856
max_pooling2d_5 (MaxPooling2D)	(None, 30, 30, 128)	0
flatten_1 (Flatten)	(None, 115200)	0
dense_2 (Dense)	(None, 128)	14,745,728
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 6)	774

```
 Total params: 14,839,750 (56.61 MB)
 Trainable params: 14,839,750 (56.61 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 305s 3s/step - accuracy: 0.2825 - loss: 1.5475 - val_accuracy: 0.1802 - val_loss: 1.4233
Epoch 2/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 292s 3s/step - accuracy: 0.3907 - loss: 1.3248 - val_accuracy: 0.2310 - val_loss: 1.3611
Epoch 3/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 301s 3s/step - accuracy: 0.5012 - loss: 1.1777 - val_accuracy: 0.4365 - val_loss: 1.4117
Epoch 4/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 290s 3s/step - accuracy: 0.5547 - loss: 1.0781 - val_accuracy: 0.4416 - val_loss: 1.2289
Epoch 5/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 332s 3s/step - accuracy: 0.5784 - loss: 0.9786 - val_accuracy: 0.4721 - val_loss: 1.1830
Epoch 6/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 292s 3s/step - accuracy: 0.6560 - loss: 0.8887 - val_accuracy: 0.4695 - val_loss: 1.1583
Epoch 7/20
```