```
import numpy as np
import pandas as pd
import os
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("uwrfkaggler/ravdess-emotional-speech-audio")

print("Path to dataset files:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/uwrfkaggler/ravdess-emotional-speech-audio?dataset_version_number=1...
100%|██████████| 429M/429M [00:09<00:00, 46.4MB/s]Extracting files...

Path to dataset files: /root/.cache/kagglehub/datasets/uwrfkaggler/ravdess-emotional-speech-audio/versions/1
```

```
# Set the path to your RAVDESS dataset directory
DATA_PATH = "/root/.cache/kagglehub/datasets/uwrfkaggler/ravdess-emotional-speech-audio/versions/1"  # Replace with your dataset path

# Prepare lists to store file paths and emotion labels
file_paths = []
emotions = []

# Extract file paths and labels from the dataset directory
for root, _, files in os.walk(DATA_PATH):
    for file in files:
        if file.endswith(".wav"):
            file_paths.append(os.path.join(root, file))
            # Extract emotion from file name format (for RAVDESS)
            emotion = int(file.split('-')[2])
            emotions.append(emotion)
```

```
# Map RAVDESS emotion labels to emotion names
emotion_map = {
    1: 'neutral',
    2: 'calm',
    3: 'happy',
    4: 'sad',
    5: 'angry',
    6: 'fearful',
    7: 'disgust',
    8: 'surprised'
}

# Convert emotion numbers to names
emotion_labels = [emotion_map[e] for e in emotions]
```

```
import librosa

# Function to extract features from audio files
def extract_features(file_path):
    # Load audio file
    audio, sample_rate = librosa.load(file_path, sr=None)

    # Extract audio features
    mfccs = np.mean(librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=40).T, axis=0)
    chroma = np.mean(librosa.feature.chroma_stft(y=audio, sr=sample_rate).T, axis=0)
    spectral_contrast = np.mean(librosa.feature.spectral_contrast(y=audio, sr=sample_rate).T, axis=0)
    zero_crossing = np.mean(librosa.feature.zero_crossing_rate(y=audio))

    # Concatenate features
    return np.concatenate([mfccs, chroma, spectral_contrast, [zero_crossing]])

# Extract features for all files
features = [extract_features(file_path) for file_path in file_paths]

# Create a DataFrame with features and labels
data_df = pd.DataFrame(features)
data_df['emotion'] = emotion_labels
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

# Encode labels
label_encoder = LabelEncoder()
data_df['emotion'] = label_encoder.fit_transform(data_df['emotion'])

# Split data
X = data_df.iloc[:, :-1].values  # Feature columns
y = to_categorical(data_df['emotion'].values)  # One-hot encoded labels
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout, LSTM

# Build the model
model = Sequential([
    Conv1D(64, kernel_size=3, strides=1, padding="same", activation="relu", input_shape=(X_train.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Dropout(0.3),
    LSTM(128, return_sequences=False),  # LSTM layer for sequence processing
    Flatten(),
    Dense(128, activation="relu"),
    Dropout(0.3),
    Dense(y_train.shape[1], activation="softmax")  # Output layer with softmax activation
])

# Compile the model
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` arg
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ conv1d_14 (Conv1D)                   │ (None, 60, 64)              │             256 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ max_pooling1d_14 (MaxPooling1D)      │ (None, 30, 64)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_21 (Dropout)                 │ (None, 30, 64)              │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ lstm (LSTM)                          │ (None, 128)                 │          98,816 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ flatten_7 (Flatten)                  │ (None, 128)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_9 (Dense)                      │ (None, 128)                 │          16,512 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_22 (Dropout)                 │ (None, 128)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_10 (Dense)                     │ (None, 8)                   │           1,032 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 116,616 (455.53 KB)
 Trainable params: 116,616 (455.53 KB)
 Non-trainable params: 0 (0.00 B)
```

```python
# Reshape data for Conv1D input
X_train = np.expand_dims(X_train, axis=2)
X_test = np.expand_dims(X_test, axis=2)

# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32, verbose=1)
```

```
Epoch 1/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 7s 52ms/step - accuracy: 0.1267 - loss: 2.0877 - val_accuracy: 0.1528 - val_loss: 2.0455
Epoch 2/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 6s 80ms/step - accuracy: 0.1709 - loss: 2.0193 - val_accuracy: 0.2899 - val_loss: 1.9308
Epoch 3/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 8s 48ms/step - accuracy: 0.2279 - loss: 1.9409 - val_accuracy: 0.2899 - val_loss: 1.9032
Epoch 4/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 7s 79ms/step - accuracy: 0.2426 - loss: 1.9358 - val_accuracy: 0.3212 - val_loss: 1.8210
Epoch 5/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 8s 47ms/step - accuracy: 0.2941 - loss: 1.8462 - val_accuracy: 0.3351 - val_loss: 1.7877
Epoch 6/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 5s 76ms/step - accuracy: 0.2858 - loss: 1.8309 - val_accuracy: 0.3438 - val_loss: 1.7292
Epoch 7/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 8s 47ms/step - accuracy: 0.3067 - loss: 1.8079 - val_accuracy: 0.3663 - val_loss: 1.7195
Epoch 8/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 7s 73ms/step - accuracy: 0.3118 - loss: 1.7740 - val_accuracy: 0.3576 - val_loss: 1.7161
Epoch 9/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 8s 47ms/step - accuracy: 0.3348 - loss: 1.7274 - val_accuracy: 0.3941 - val_loss: 1.6032
Epoch 10/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - accuracy: 0.3632 - loss: 1.6862 - val_accuracy: 0.3837 - val_loss: 1.6318
Epoch 11/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step - accuracy: 0.3238 - loss: 1.7040 - val_accuracy: 0.4062 - val_loss: 1.5936
Epoch 12/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 7s 77ms/step - accuracy: 0.3621 - loss: 1.6532 - val_accuracy: 0.3872 - val_loss: 1.5862
Epoch 13/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 4s 49ms/step - accuracy: 0.3970 - loss: 1.6393 - val_accuracy: 0.4167 - val_loss: 1.5684
Epoch 14/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - accuracy: 0.4002 - loss: 1.5901 - val_accuracy: 0.4184 - val_loss: 1.5376
Epoch 15/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 6s 65ms/step - accuracy: 0.3886 - loss: 1.5853 - val_accuracy: 0.4722 - val_loss: 1.4695
Epoch 16/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 4s 48ms/step - accuracy: 0.4172 - loss: 1.5439 - val_accuracy: 0.4670 - val_loss: 1.4497
Epoch 17/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - accuracy: 0.4295 - loss: 1.5313 - val_accuracy: 0.5000 - val_loss: 1.4474
```

```
Epoch 18/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 4s 50ms/step - accuracy: 0.4433 - loss: 1.4943 - val_accuracy: 0.4618 - val_loss: 1.4253
Epoch 19/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - accuracy: 0.4635 - loss: 1.4621 - val_accuracy: 0.5069 - val_loss: 1.3927
Epoch 20/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 7s 80ms/step - accuracy: 0.4777 - loss: 1.4194 - val_accuracy: 0.4531 - val_loss: 1.4116
Epoch 21/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 8s 50ms/step - accuracy: 0.4576 - loss: 1.4693 - val_accuracy: 0.4774 - val_loss: 1.4071
Epoch 22/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 7s 76ms/step - accuracy: 0.5008 - loss: 1.3516 - val_accuracy: 0.4983 - val_loss: 1.3251
Epoch 23/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 8s 48ms/step - accuracy: 0.5065 - loss: 1.3684 - val_accuracy: 0.5087 - val_loss: 1.3574
Epoch 24/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 6s 61ms/step - accuracy: 0.4939 - loss: 1.3486 - val_accuracy: 0.5295 - val_loss: 1.2910
Epoch 25/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 4s 48ms/step - accuracy: 0.4978 - loss: 1.3244 - val_accuracy: 0.5174 - val_loss: 1.3064
Epoch 26/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.5150 - loss: 1.3076 - val_accuracy: 0.5122 - val_loss: 1.3097
Epoch 27/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 6s 60ms/step - accuracy: 0.5221 - loss: 1.2910 - val_accuracy: 0.5365 - val_loss: 1.2839
Epoch 28/50
72/72 ━━━━━━━━━━━━━━━━━━━━ 4s 49ms/step - accuracy: 0.5127 - loss: 1.2996 - val_accuracy: 0.5174 - val_loss: 1.2786
Epoch 29/50
```

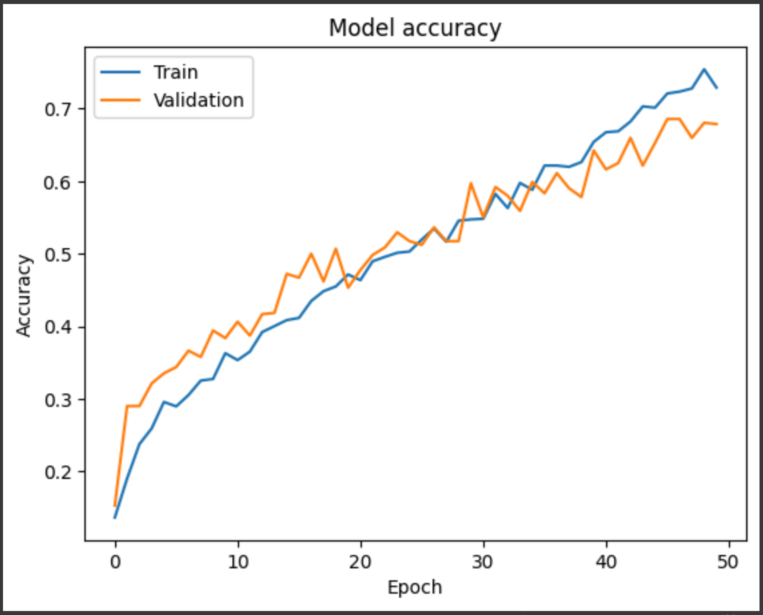
```python
# Evaluate model performance
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.6669 - loss: 0.8591
Test Accuracy: 67.88%
```

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Save the model
model.save("emotion_recognition_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 lega
```

Start coding or generate with AI.