This code is part of your **Speech Emotion Recognition (SER) system**, which aims to classify emotions from speech recordings. Let’s break it down step by step in simple terms.

**1. Overview of the Code**

This code processes an audio file, extracts **Mel spectrogram** features, applies **deep learning** (CNN + LSTM model), and predicts the speaker’s **emotion**.

It involves:

* **Extracting features** from audio using Mel spectrograms.
* **Dividing the spectrogram** into smaller parts (chunks) for better learning.
* **Using a deep learning model** (CNN + LSTM) to classify the emotion.

**2. Importing Required Libraries**

The code imports various libraries for:

* **Audio processing:** librosa (extracting Mel spectrograms).
* **Deep learning:** torch (PyTorch for CNN + LSTM model).
* **Data scaling:** sklearn.preprocessing.StandardScaler (to normalize audio features).

**3. Feature Extraction - Mel Spectrogram**

**Function: getMELspectrogram(audio, sample\_rate)**

🔹 **Purpose:** Converts raw audio into a **Mel spectrogram**, which is an image-like representation of sound.

**Steps:**

1. **Mel spectrogram generation** → Uses librosa.feature.melspectrogram().
2. **Convert to decibels (dB)** → Uses librosa.power\_to\_db().
3. **Returns the Mel spectrogram** (which will be input for the model).

📌 **Why use a Mel spectrogram?**

* Converts speech into a visual form that the **CNN model** can process.
* Mimics human hearing, focusing on important sound frequencies.

**4. Splitting Audio Data into Chunks**

**Function: splitIntoChunks(mel\_spec, win\_size, stride)**

🔹 **Purpose:** Splits the Mel spectrogram into **smaller overlapping parts** (chunks).

**Steps:**

1. Calculates the **number of chunks** possible.
2. Iterates through the spectrogram, extracting small **windows** of size win\_size.
3. Stores only chunks of the exact win\_size.
4. Returns the stacked chunks as a NumPy array.

📌 **Why split into chunks?**

* **LSTMs require sequential data** → Chunks allow learning temporal patterns.
* **CNNs work better with small patches** → Instead of analyzing the whole spectrogram at once, CNN processes chunks separately.

**5. TimeDistributed Wrapper for CNN Layers**

**Class: TimeDistributed(nn.Module)**

🔹 **Purpose:** Applies the **same layer** (like Conv2D) to each time step in the spectrogram **separately**.

📌 **Why use TimeDistributed?**

* **CNN extracts features for each time step independently.**
* **LSTM processes features over time.**
* Helps **combine CNN (image processing) with LSTM (sequential learning)**.

**6. Hybrid Model (CNN + LSTM)**

**Class: HybridModel(nn.Module)**

🔹 **Purpose:** This is the deep learning model that takes the Mel spectrogram chunks and predicts **emotion**.

📌 **Model Architecture:**

* **Convolutional layers (CNN)** extract spatial (image-like) features.
* **LSTM layer** learns how these features change over time.
* **Fully connected layer (Linear layer)** classifies emotion.
* **Softmax activation** gives probabilities for each emotion.

**How the model works step by step?**

1. **CNN layers** process each time step's Mel spectrogram to extract **deep features**.
2. **LSTM layer** takes CNN features and **learns sequential patterns**.
3. **Final layer** maps LSTM outputs to **emotion classes**.

📌 **Why use CNN + LSTM?**

* **CNN captures spatial features (like patterns in sound).**
* **LSTM captures time-based relationships (like emotion progression).**
* **Combining both improves emotion classification accuracy.**

**7. Loading the Pretrained Model**

model = HybridModel(len(EMOTIONS))

model.load\_state\_dict(torch.load("/workspaces/SER\_/models/cnn\_lstm\_model\_imp1.pt", map\_location=torch.device('cpu')))

🔹 Loads a **pre-trained deep learning model** (saved as .pt file) and prepares it for inference.

📌 **Why load a pre-trained model?**

* The model was **already trained** on an emotion dataset.
* We only need to **use it for prediction** (no training required).

**8. Preprocessing the Audio for Prediction**

**Function: process\_files(audio\_path)**

🔹 **Purpose:** Loads an audio file, extracts features, processes them, and predicts the **emotion**.

**Step-by-Step Breakdown**

1. **Loads the audio file** using librosa.load().
   * Converts it into a fixed-size signal of **3 seconds**.
2. **Extracts Mel spectrograms** using getMELspectrogram().
3. **Splits the spectrogram into chunks** using splitIntoChunks().
4. **Reshapes and normalizes the data** using StandardScaler().
5. **Converts the processed data into a PyTorch tensor** for input to the model.
6. **Runs the model for prediction.**
7. **Finds the most likely emotion** and returns the corresponding **emoji + label**.

📌 **Why standardize the data (StandardScaler)?**

* **Ensures consistent scale across different audio files.**
* **Improves model performance** by making features more uniform.

**9. Making Predictions**

model.eval()

output\_logits, output\_softmax = model(X\_tensor)

predictions = torch.argmax(output\_softmax, dim=1).tolist()[0]

return EMOTIONS[predictions]

🔹 Runs the model in **evaluation mode** (no training).  
🔹 Gets **probabilities** for each emotion.  
🔹 Chooses the emotion with the **highest probability**.

📌 **Why use softmax?**

* Converts raw model outputs into probabilities.
* Helps in selecting the most **confident** emotion prediction.

**10. Emotion Labels and Emojis**

EMOTIONS = {1:'neutral 😐', 2:'calm 😌', 3:'happy 😁', 4:'sad 😟',

5:'angry 😠', 6:'fear 😨', 7:'disgust 🤢', 0:'surprise 😮'}

🔹 Maps **predicted class IDs** to **human-readable emotions with emojis**.  
🔹 Example: If predictions = 3, it returns **"happy 😁"**.

**Summary of What Happens in This Code**

1. **Loads an audio file.**
2. **Extracts a Mel spectrogram** (image-like sound representation).
3. **Splits the spectrogram into smaller chunks** for better feature learning.
4. **Passes the chunks through a CNN + LSTM model** to extract meaningful features.
5. **Predicts the most probable emotion.**
6. **Returns the emotion with an emoji.**

**Final Thoughts**

This code is **a complete deep learning pipeline for speech emotion recognition**. The **CNN layers** extract features from the Mel spectrogram, and the **LSTM layers** analyze time-based patterns. Finally, the model predicts the **emotion from the voice** and returns an emoji-based label.

Let me know if you need a specific part explained in more detail! 😊

I will now explain your code **line by line** in **simple terms**, covering what each line does, why it's used, and how it fits into the **Speech Emotion Recognition (SER) system**.

**1. Importing Required Libraries**

**🔹 Code:**

import numpy as np

import librosa

import librosa.display

import torch

import torch.nn as nn

from sklearn.preprocessing import StandardScaler

**📌 What This Does?**

* **numpy** → Used for numerical operations, array handling, and mathematical computations.
* **librosa** → A Python library for audio processing:
  + librosa.load() loads audio files.
  + librosa.feature.melspectrogram() extracts Mel spectrograms.
* **torch** → PyTorch framework for deep learning.
* **torch.nn** → Provides layers like CNN, LSTM, and activation functions.
* **StandardScaler** → Standardizes features (mean = 0, variance = 1) for stable training.

**2. Function: getMELspectrogram(audio, sample\_rate)**

**🔹 Code:**

def getMELspectrogram(audio, sample\_rate):

mel\_spec = librosa.feature.melspectrogram(y=audio,

sr=sample\_rate,

n\_fft=1024,

win\_length=512,

window='hamming',

hop\_length=256,

n\_mels=128,

fmax=sample\_rate / 2

)

mel\_spec\_db = librosa.power\_to\_db(mel\_spec, ref=np.max)

return mel\_spec\_db

**📌 What This Does?**

1. **Converts raw audio into a Mel spectrogram** (a time-frequency representation of sound).
2. **Breakdown of Parameters:**
   * n\_fft=1024 → Number of **Fourier Transform points** (splits audio into frequencies).
   * win\_length=512 → Size of each window for Fourier Transform.
   * window='hamming' → Applies a **Hamming window** (smoothes frequency transitions).
   * hop\_length=256 → Step size for moving the window (overlapping to maintain detail).
   * n\_mels=128 → Converts sound into **128 Mel frequency bands** (human ear perception).
   * fmax=sample\_rate / 2 → Keeps only frequencies **up to half the sample rate**.
3. **librosa.power\_to\_db()** converts power to decibels (makes spectrogram readable).
4. **Returns the processed Mel spectrogram** to be fed into the CNN.

**3. Function: splitIntoChunks(mel\_spec, win\_size, stride)**

**🔹 Code:**

def splitIntoChunks(mel\_spec, win\_size, stride):

t = mel\_spec.shape[1]

num\_of\_chunks = int(t / stride)

chunks = []

for i in range(num\_of\_chunks):

chunk = mel\_spec[:, i \* stride:i \* stride + win\_size]

if chunk.shape[1] == win\_size:

chunks.append(chunk)

return np.stack(chunks, axis=0)

**📌 What This Does?**

1. **Divides the Mel spectrogram into smaller overlapping chunks.**
2. **Why?** CNNs and LSTMs work better when analyzing smaller time frames.
3. **Key parts of the function:**
   * t = mel\_spec.shape[1] → Gets the number of **time frames** in the spectrogram.
   * num\_of\_chunks = int(t / stride) → Determines how many chunks we can extract.
   * **Loop:** Extracts **small windows (win\_size=128) with some overlap (stride=64)**.
   * **Only keeps chunks with the exact size.**

📌 **Example:**

* If Mel spectrogram has **1000 time frames**, and win\_size=128, stride=64 →  
  We get **around 15 overlapping chunks**.

**4. TimeDistributed Layer (Custom PyTorch Module)**

**🔹 Code:**

class TimeDistributed(nn.Module):

def \_\_init\_\_(self, module):

super(TimeDistributed, self).\_\_init\_\_()

self.module = module

def forward(self, x):

if len(x.size()) <= 2:

return self.module(x)

elif len(x.size()) == 3:

x\_reshape = x.contiguous().view(-1, x.size(2))

elif len(x.size()) == 4:

x\_reshape = x.contiguous().view(-1, x.size(2), x.size(3))

else:

x\_reshape = x.contiguous().view(-1, x.size(2), x.size(3), x.size(4))

y = self.module(x\_reshape)

if len(x.size()) == 3:

y = y.contiguous().view(x.size(0), -1, y.size(1))

elif len(x.size()) == 4:

y = y.contiguous().view(x.size(0), -1, y.size(1), y.size(2))

else:

y = y.contiguous().view(x.size(0), -1, y.size(1), y.size(2), y.size(3))

return y

**📌 What This Does?**

1. **Wraps CNN layers to apply them to each time step separately.**
2. **Why?** CNN extracts features for **each time step independently** before LSTM processes sequences.
3. **Reshapes input (view())** to apply layers to each time step in batch mode.

**5. Hybrid CNN-LSTM Model**

**🔹 Code:**

class HybridModel(nn.Module):

def \_\_init\_\_(self, num\_emotions):

super().\_\_init\_\_()

self.conv2Dblock = nn.Sequential(

TimeDistributed(nn.Conv2d(1, 16, 3, 1, 1)),

TimeDistributed(nn.BatchNorm2d(16)),

TimeDistributed(nn.ReLU()),

TimeDistributed(nn.MaxPool2d(2, 2)),

TimeDistributed(nn.Dropout(0.4)),

TimeDistributed(nn.Conv2d(16, 32, 3, 1, 1)),

TimeDistributed(nn.BatchNorm2d(32)),

TimeDistributed(nn.ReLU()),

TimeDistributed(nn.MaxPool2d(4, 4)),

TimeDistributed(nn.Dropout(0.4)),

TimeDistributed(nn.Conv2d(32, 64, 3, 1, 1)),

TimeDistributed(nn.BatchNorm2d(64)),

TimeDistributed(nn.ReLU()),

TimeDistributed(nn.MaxPool2d(4, 4)),

TimeDistributed(nn.Dropout(0.4)),

TimeDistributed(nn.Conv2d(64, 128, 3, 1, 1)),

TimeDistributed(nn.BatchNorm2d(128)),

TimeDistributed(nn.ReLU()),

TimeDistributed(nn.MaxPool2d(4, 4)),

TimeDistributed(nn.Dropout(0.4))

)

self.lstm = nn.LSTM(128, 64, bidirectional=False, batch\_first=True)

self.dropout\_lstm = nn.Dropout(0.3)

self.out\_linear = nn.Linear(64, num\_emotions)

def forward(self, x):

conv\_embedding = self.conv2Dblock(x)

conv\_embedding = torch.flatten(conv\_embedding, start\_dim=2)

lstm\_embedding, (h, c) = self.lstm(conv\_embedding)

lstm\_embedding = self.dropout\_lstm(lstm\_embedding)

lstm\_output = lstm\_embedding[:, -1, :]

output\_logits = self.out\_linear(lstm\_output)

output\_softmax = nn.functional.softmax(output\_logits, dim=1)

return output\_logits, output\_softmax

**📌 What This Does?**

1. **CNN extracts spatial features** from Mel spectrograms.
2. **LSTM learns time-based changes** in emotions.
3. **Final Linear layer maps features to emotions.**
4. **Softmax activation** produces probabilities for each emotion.

This is the **detailed line-by-line explanation** of your code. Let me know if you need further breakdowns! 😊

I will now **explain each line** of the HybridModel class **in detail**, breaking it down **step by step** to help you understand its functionality.

**📌 Class Definition**

**Code:**

class HybridModel(nn.Module):

**What This Does?**

* **Defines a PyTorch model** named HybridModel that combines **CNN** and **LSTM** architectures.
* **Inherits from nn.Module**, making it a PyTorch neural network model.

**📌 Constructor (\_\_init\_\_ method)**

**Code:**

def \_\_init\_\_(self, num\_emotions):

**What This Does?**

* This is the **initializer (constructor)** method.
* **num\_emotions** → Number of emotion classes (e.g., happy, sad, angry, etc.).
* This value is used in the final output layer.

**Code:**

super().\_\_init\_\_()

**What This Does?**

* Calls the **parent class (nn.Module) constructor**.
* This is necessary for **PyTorch models** to work correctly.

**📌 1️⃣ CNN Block (Feature Extraction)**

**Code:**

self.conv2Dblock = nn.Sequential(

**What This Does?**

* Defines a **CNN feature extraction block** using nn.Sequential().
* This allows us to apply multiple **CNN layers** in a sequence.

**🔹 First Convolution Block**

**Code:**

TimeDistributed(nn.Conv2d(1, 16, 3, 1, 1)),

TimeDistributed(nn.BatchNorm2d(16)),

TimeDistributed(nn.ReLU()),

TimeDistributed(nn.MaxPool2d(2, 2)),

TimeDistributed(nn.Dropout(0.4)),

**What This Does?**

| **Layer** | **Function** |
| --- | --- |
| Conv2d(1, 16, 3, 1, 1) | **Convolutional layer** (1 input channel, 16 output filters, kernel=3x3, stride=1, padding=1). |
| BatchNorm2d(16) | **Batch normalization** to stabilize training and improve performance. |
| ReLU() | **Activation function** (ReLU) introduces non-linearity. |
| MaxPool2d(2, 2) | **Reduces** feature map size by **half** (2x2 pooling). |
| Dropout(0.4) | **Reduces overfitting** by randomly disabling 40% of neurons. |

🔹 **Why TimeDistributed?**

* It **applies CNN to each time frame separately**, treating them as independent 2D images.

**🔹 Second Convolution Block**

**Code:**

TimeDistributed(nn.Conv2d(16, 32, 3, 1, 1)),

TimeDistributed(nn.BatchNorm2d(32)),

TimeDistributed(nn.ReLU()),

TimeDistributed(nn.MaxPool2d(4, 4)),

TimeDistributed(nn.Dropout(0.4)),

**Changes from First Block:**

* Input: 16 filters → Output: 32 filters (increased complexity).
* Pooling window changed to **4x4** (further reduces feature size).

**🔹 Third Convolution Block**

**Code:**

TimeDistributed(nn.Conv2d(32, 64, 3, 1, 1)),

TimeDistributed(nn.BatchNorm2d(64)),

TimeDistributed(nn.ReLU()),

TimeDistributed(nn.MaxPool2d(4, 4)),

TimeDistributed(nn.Dropout(0.4)),

**Changes from Second Block:**

* Input: 32 filters → Output: 64 filters.
* Further increases **feature extraction ability**.

**🔹 Fourth Convolution Block**

**Code:**

TimeDistributed(nn.Conv2d(64, 128, 3, 1, 1)),

TimeDistributed(nn.BatchNorm2d(128)),

TimeDistributed(nn.ReLU()),

TimeDistributed(nn.MaxPool2d(4, 4)),

TimeDistributed(nn.Dropout(0.4))

**Changes from Third Block:**

* Input: 64 filters → Output: 128 filters.
* Final CNN layer **extracts high-level features**.

**📌 2️⃣ LSTM Block (Sequence Learning)**

**Code:**

self.lstm = nn.LSTM(128, 64, bidirectional=False, batch\_first=True)

**What This Does?**

* **LSTM (Long Short-Term Memory)** network processes **sequential time-dependent features**.
* **Input size: 128** (output from CNN feature maps).
* **Hidden size: 64** (reduces feature dimensions).
* **bidirectional=False** → Uses a single LSTM direction (not forward-backward).
* **batch\_first=True** → Input is in the format **(batch, time, features)**.

**Code:**

self.dropout\_lstm = nn.Dropout(0.3)

**What This Does?**

* Adds **dropout (30%)** to prevent overfitting.

**📌 3️⃣ Fully Connected Layer (Emotion Classification)**

**Code:**

self.out\_linear = nn.Linear(64, num\_emotions)

**What This Does?**

* **Maps LSTM output (64 features) to emotion classes (num\_emotions).**
* If there are **8 emotions**, this layer outputs **8 values**.

**📌 Forward Pass (Processing Input Data)**

**Code:**

def forward(self, x):

**What This Does?**

* Defines how the model **processes input audio features**.

**Step 1: Pass Input Through CNN**

**Code:**

conv\_embedding = self.conv2Dblock(x)

**What This Does?**

* Passes input x (Mel spectrogram) **through CNN layers**.
* Output: conv\_embedding → **Extracted feature maps**.

**Step 2: Flatten CNN Output**

**Code:**

conv\_embedding = torch.flatten(conv\_embedding, start\_dim=2)

**What This Does?**

* Converts CNN output **from 4D (batch, time, channels, height, width) to 3D (batch, time, features)**.
* LSTM **only accepts 3D input**.

**Step 3: Pass Data Through LSTM**

**Code:**

lstm\_embedding, (h, c) = self.lstm(conv\_embedding)

**What This Does?**

* Passes CNN features **through LSTM** for sequential learning.
* **LSTM output:**
  + lstm\_embedding → Processed sequence output.
  + h, c → LSTM’s hidden and cell states (not used in this case).

**Step 4: Apply Dropout to LSTM Output**

**Code:**

lstm\_embedding = self.dropout\_lstm(lstm\_embedding)

**What This Does?**

* Reduces overfitting **by randomly dropping neurons**.

**Step 5: Extract Final Time Step Output**

**Code:**

lstm\_output = lstm\_embedding[:, -1, :]

**What This Does?**

* Extracts **the last time step's output** from the LSTM sequence.
* Represents the **final learned emotion features**.

**Step 6: Fully Connected Layer**

**Code:**

output\_logits = self.out\_linear(lstm\_output)

**What This Does?**

* Passes LSTM features **through the final classification layer**.
* Outputs **raw emotion scores (logits)**.

**Step 7: Apply Softmax (Convert to Probabilities)**

**Code:**

output\_softmax = nn.functional.softmax(output\_logits, dim=1)

**What This Does?**

* Converts raw scores into **probabilities for each emotion class**.

**Step 8: Return Outputs**

**Code:**

return output\_logits, output\_softmax

**What This Does?**

* Returns **two values**:
  + output\_logits → Raw emotion scores.
  + output\_softmax → Probabilities for each emotion.

**🔹 Summary of Model**

✔ CNN extracts **spatial features** from Mel spectrograms.  
✔ LSTM learns **temporal patterns** in speech.  
✔ Fully connected layer **predicts the emotion**.

Let me know if you need any clarifications! 🚀😊