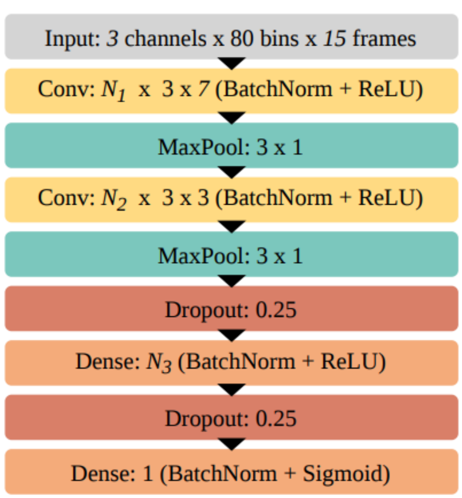
Stroke detection

Preprocessing:

- Audio Feature Extraction:
    - Spectrograms: Transform audio signals into spectrograms, providing a visual representation of frequency content over time.
    - Mel-Frequency Cepstral Coefficients (MFCCs): Extract coefficients representing the short-term power spectrum of sound, capturing crucial audio features.
    - Chroma Features: Describe pitch class energy distribution, useful for capturing tonal content.
- Data Augmentation:
    - Apply techniques like random pitch shifting, time stretching, and background noise addition to the training set for enhanced model robustness.

Model Architecture:

- Convolutional Neural Network (CNN) for Tabla Strokes:
    - Input Layer: Accept preprocessed audio features (e.g., spectrograms, MFCCs, chroma features).
    - Convolutional Layers: Extract hierarchical features, capturing patterns and relationships in audio features.
    - Pooling Layers: Downsample spatial dimensions, reducing computational load and enhancing translational invariance.
    - Flatten Layer: Flatten output from convolutional layers for input to fully connected layers.
    - Fully Connected Layers: Make predictions based on learned features, with the final layer having four nodes for tabla stroke categories.
    - Activation Functions: Use ReLU or similar functions to introduce non-linearity.



Transfer Learning:

- Source Domain (Western Drums):
    - Pretraining:
        - Utilize a dataset containing Western drum sounds.
        - Train initial CNN layers to capture general features related to drum sounds, initializing the network with cross-domain knowledge.
- Target Domain (Tabla Strokes):
    - Fine-Tuning:
        - Use the pretrained model as a starting point.
        - Replace or fine-tune later layers to adapt the model to tabla strokes.
        - Train the model on the tabla dataset to learn specific tabla stroke features.
        - Leverage knowledge gained from the source domain while adapting to tabla stroke nuances.

This holistic approach, combining detailed audio feature extraction, a robust CNN architecture, and strategic transfer learning, addresses challenges related to limited tabla stroke data and exploits similarities with Western drum sounds for improved model performance.

Basic code

```
import librosa

import librosa.display

import matplotlib.pyplot as plt

import numpy as np

import os

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder
```

```
from tensorflow.keras.utils import to_categorical


# Function to load data

def load_data(data_path, labels_path)

    features = []

    labels = []

    for audio_file in os.listdir(data_path):

        if audio_file.endswith(".wav"):

            audio_path = os.path.join(data_path, audio_file)

            # Extract label from filename or use a separate file for labels

            label = get_label_from_filename(audio_file)

            labels.append(label)


            # Load audio file and extract features

            audio_data, _ = librosa.load(audio_path, sr=None)

            extracted_features = extract_features(audio_data)

            features.append(extracted_features)
```

```
    return np.array(features), np.array(labels)


# Function to extract features from audio data
def extract_features(audio_data):
    # Using Mel-Frequency Cepstral Coefficients (MFCCs) as features
    mfccs = librosa.feature.mfcc(y=audio_data, sr=44100, n_mfcc=13)
    return mfccs.flatten()


# Function to apply data augmentation
def augment_data(train_features, train_labels):
    augmented_features = []
    augmented_labels = []

    for i in range(len(train_features)):
            augmented_audio = apply_augmentation(train_features[i])
        # Extract features from augmented audio
        augmented_features.append(extract_features(augmented_audio))
        augmented_labels.append(train_labels[i])
```

```
    return np.array(augmented_features), np.array(augmented_labels)


# Function to apply random pitch shifting for data augmentation
def apply_augmentation(audio_data, sr=44100, pitch_shift_steps=2):
    # Randomly shift pitch within the specified number of steps
    pitch_shift_amount = np.random.uniform(-pitch_shift_steps, pitch_shift_steps)
    # Apply pitch shifting
    augmented_audio = librosa.effects.pitch_shift(audio_data, sr, n_steps=pitch_shift_amount)
    return augmented_audio
# Function to get label from filename (assuming filename is in the format "label_filename.wav")
def get_label_from_filename(filename):
    return filename.split("_")[0]
data_path = "path/to/audio/files"
labels_path = "path/to/labels"
features, labels = load_data(data_path, labels_path)


# Converting labels to one-hot encoding
```

label_encoder = LabelEncoder()

encoded_labels = label_encoder.fit_transform(labels)

one_hot_labels = to_categorical(encoded_labels)

Split the dataset into training and testing sets

train_features, test_features, train_labels, test_labels = train_test_split(features, one_hot_labels, test_size=0.2, random_state=42)

Model

import tensorflow as tf

from tensorflow.keras import layers, models

def create_tabla_model(input_shape):

    model = models.Sequential()

    # Input Layer

    model.add(layers.InputLayer(input_shape=input_shape))

    # Convolutional Layers

    model.add(layers.Conv2D(32, (3, 3), activation='relu'))

    model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Conv2D(64, (3, 3), activation='relu'))

```
    model.add(layers.MaxPooling2D((2, 2)))

    model.add(layers.Conv2D(128, (3, 3), activation='relu'))

    model.add(layers.MaxPooling2D((2, 2)))

    #Flatten Layer

    model.add(layers.Flatten())

    # Fully Connected Layers

    model.add(layers.Dense(128, activation='relu'))

    model.add(layers.Dropout(0.5))

    model.add(layers.Dense(4, activation='softmax'))  # 4 output nodes for the four tabla stroke
categories

    # Compile the model

    model.compile(optimizer='adam',

                  loss='sparse_categorical_crossentropy',

                  metrics=['accuracy'])



    return model

# Assuming input_shape is the shape of your preprocessed audio features

input_shape = (num_frames, num_features, 1)  # Update with your actual dimensions
```

```python
tabla_model = create_tabla_model(input_shape)

# Display the model summary

tabla_model.summary()
```