**Comparative analysis of ML and DL models**

**Loading audio files:**

The below code loads audio data from a main folder called sort\_data, where each subfolder represents a different class. For each audio file in these subfolders, the code extracts features (using the extract\_features function) and stores them in the features list. The corresponding class labels (the names of the subfolders) are stored in the labels list. This prepares the data for further processing or model training.

|  |
| --- |
| **# Define the path to your main folder**  **main\_folder\_path = 'sort\_data'**  **features, labels = [], []**  **# Load data and extract features**  **for class\_folder in os.listdir(main\_folder\_path):**  **class\_path = os.path.join(main\_folder\_path, class\_folder)**  **if os.path.isdir(class\_path):**  **for file\_name in os.listdir(class\_path):**  **file\_path = os.path.join(class\_path, file\_name)**  **features.append(extract\_features(file\_path))**  **labels.append(class\_folder)** |

**Extracting features:**

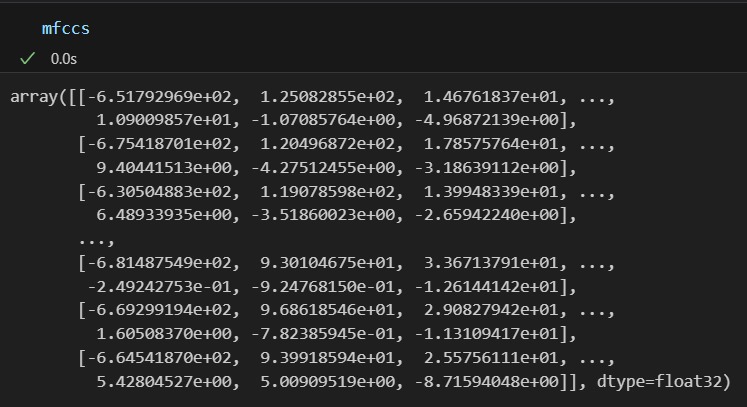
The extract\_features function loads an audio file and extracts mel spectrogram features. It computes the mel spectrogram, averages it over time to reduce dimensionality, and returns the mean values as the feature representation for the audio file.

|  |
| --- |
| def extract\_features(file\_path):  y, sr = librosa.load(file\_path, sr=None)  mel = librosa.feature.melspectrogram(y=y, sr=sr, n\_fft = 2048, hop\_length=512, n\_mels=10)  mel\_mean = np.mean(mel.T, axis=0)  return mel\_mean |

**Converting lists to numpy arrays:**

The code converts the features and labels lists into NumPy arrays. This makes it easier to handle and manipulate the data, particularly for tasks like model training, where operations on arrays are more efficient and commonly required.

|  |
| --- |
| **# Convert features and labels to numpy arrays for easier handling**  **features = np.array(features)**  **labels = np.array(labels)** |



**Numpy array after extracting MFCC from raw audio signal**

**Feature-specific model training for audio classification:**

**The codes given below train different models for different features, lets see one by one in detail:**

**Features: Spectrogram and Mel-spectrogram**

1. **Random Forest Classifier:**

The model is initialized with 100 decision trees (n\_estimators=100) and a fixed random seed (random\_state=42) to ensure reproducibility.

|  |
| --- |
| # Train a Random Forest Classifier  model = RandomForestClassifier(n\_estimators=100, random\_state=42)  model.fit(X\_train, y\_train) |

1. Accuracy for Mel-spectrogram: 88%
2. Accuracy of spectrogram: 76.71%
3. Accuracy of Mel-spectrogram + MFCC: 96.86%

**2. K-Nearest Neighbor:**

|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier  knn = KNeighborsClassifier(n\_neighbors = 5)  knn.fit(X\_train,y\_train) |

1. Accuracy for Mel-spectrogram: 54.46%
2. Accuracy of spectrogram: 96.58%
3. Accuracy of Mel-spectrogram + MFCC: 95.05%

**3. CNN:**

The load\_custom\_cnn\_model function defines a simple Convolutional Neural Network (CNN) model designed to classify grayscale images of size 128x128 into 10 classes. The model consists of two convolutional layers with 32 and 64 filters, respectively, each followed by max pooling to downsample the feature maps. After flattening the feature maps into a 1D vector, the model has a dense layer with 128 neurons for further processing, followed by a dropout layer to prevent overfitting. The final output layer uses softmax activation to produce probabilities for the 10 classes. This architecture efficiently extracts hierarchical features from the input images, making it suitable for image classification tasks.

* Spectrogram: 99.4%
* Mel Spectrogram: 99.2%

|  |
| --- |
| def load\_custom\_cnn\_model(input\_shape):  model = tf.keras.Sequential([  tf.keras.layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(128, 128, 1)),  tf.keras.layers.MaxPooling2D(pool\_size=(2, 2)),  tf.keras.layers.Conv2D(64, kernel\_size=(3, 3), activation='relu'),  tf.keras.layers.MaxPooling2D(pool\_size=(2, 2)),  tf.keras.layers.Flatten(),  tf.keras.layers.Dense(128, activation='relu'),  tf.keras.layers.Dropout(0.5),  tf.keras.layers.Dense(10, activation='softmax')  ])  return model |

**4. LSTM**:

The load\_custom\_lstm\_model function defines a simple Long Short-Term Memory (LSTM) network tailored for sequence classification tasks. The model begins with an LSTM layer containing 64 units, designed to capture temporal dependencies in sequential data, such as time-series or audio features. This is followed by a dense layer with 128 neurons and ReLU activation to further process the extracted features. A dropout layer is included to mitigate overfitting by randomly dropping 50% of the neurons during training. Finally, a softmax layer outputs probabilities across 10 classes, making this model suitable for tasks that involve classifying sequences into predefined categories.

* Spectrogram: 96.3%
* Mel Spectrogram:95.3%

|  |
| --- |
| **def load\_custom\_lstm\_model(input\_shape):**  **model = tf.keras.Sequential([**  **tf.keras.layers.LSTM(64, input\_shape=(128, 128)),**  **tf.keras.layers.Dense(128, activation='relu'),**  **tf.keras.layers.Dropout(0.5),**  **tf.keras.layers.Dense(10, activation='softmax')**  **])**  **return model** |

**5. CNN + LSTM:**

The load\_cnn\_lstm\_combination function creates a hybrid model that combines Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to leverage both spatial and temporal features. The model starts with two convolutional layers (with 32 and 64 filters respectively) that capture spatial patterns in the input data, followed by max-pooling layers to reduce dimensionality. Afterward, the output is flattened using a TimeDistributed layer, allowing the subsequent LSTM layer to process the temporal sequence of features. The LSTM layer with 64 units captures temporal dependencies, and the final dense layers, including a dropout for regularization, conclude with a softmax layer to output probabilities for 10 classes. This combination is particularly effective for tasks like audio or video classification, where both spatial and temporal information are crucial.

* Spectrogram: 99.1%
* Mel Spectrogram: 98.6%

|  |
| --- |
| **def load\_cnn\_lstm\_combination(input\_shape):**  **model = tf.keras.Sequential([**  **tf.keras.layers.Conv2D(32, kernel\_size=(3, 3), activation='relu', input\_shape=(input\_shape[0], input\_shape[1], 1)),**  **tf.keras.layers.MaxPooling2D(pool\_size=(2, 2)),**  **tf.keras.layers.Conv2D(64, kernel\_size=(3, 3), activation='relu'),**  **tf.keras.layers.MaxPooling2D(pool\_size=(2, 2)),**  **tf.keras.layers.TimeDistributed(tf.keras.layers.Flatten()),**  **tf.keras.layers.LSTM(64),**  **tf.keras.layers.Dense(128, activation='relu'),**  **tf.keras.layers.Dropout(0.5),**  **tf.keras.layers.Dense(10, activation='softmax')**  **])**  **return model** |

**Features: MFCC and Chroma**

1. **Random Forest Classifier:**

|  |
| --- |
| # Train a Random Forest Classifier  model = RandomForestClassifier(n\_estimators=100, random\_state=42)  model.fit(X\_train, y\_train) |

* Accuracy for MFCC: 100 %
* Accuracy of Chroma CENS: 38.38%
* Accuracy of Chroma + MFCC: 94%

1. **K-Nearest Neighbor:**

|  |
| --- |
| **from sklearn.neighbors import KNeighborsClassifier**  **knn = KNeighborsClassifier(n\_neighbors = 5)**  **knn.fit(X\_train,y\_train)** |

* Accuracy for MFCC: 89.01 %
* Accuracy of Chroma CENS: 34.61%
* Accuracy of Chroma + MFCC: 95.06%

1. **Lstm :**

The given code defines, trains, and evaluates LSTM models for audio classification using both MFCC and Chroma features. We begin by building an LSTM model with 64 units in the first LSTM layer, a Dropout layer for regularization, 32 units in the second LSTM layer, and a final Dense layer with 10 units for classification with a softmax activation function. The model is compiled with the Adam optimizer and sparse categorical crossentropy loss. The training data, reshaped for LSTM input, is used to fit the model for 10 epochs with a 10% validation split. After training, the model is evaluated on the test set. The accuracies are printed, showing that the model achieved an accuracy of 73.97% for MFCC features but only 21.52% for Chroma features, indicating that MFCCs were more effective for this classification task.

We compute their means over time, and concatenate them into a single feature vector. The LSTM model trained with these features achieves a test accuracy of 74.85%, and for the normalized values of the mfccs and chroma feature we have got accuracies of 98.20% and 66.02% respectively.

|  |
| --- |
| model = Sequential()  model.add(LSTM(64, return\_sequences=True, input\_shape=input\_shape))  model.add(Dropout(0.5))  model.add(LSTM(32))  model.add(Dense(10, activation='softmax'))  model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy']) |

1. **CNN:**

The code defines and evaluates Convolutional Neural Networks (CNNs) for classifying audio features extracted from MFCC and Chroma. The CNN model, which includes layers for convolution, pooling, and dense classification, is first trained on MFCC features, achieving an accuracy of 82.57%. It is then trained on Chroma features, yielding a lower accuracy of 27.22%. This comparison highlights that MFCC features provide more effective input for the CNN model in this classification task compared to Chroma features.

We compute their means over time, and concatenate them into a single feature vector. This approach achieves a test accuracy of 87.63% and for the normalized values of the mfccs and chroma feature we have got accuracies of 99.15% and 72.83% respectively.

|  |
| --- |
| model = Sequential()  model.add(Conv1D(32, kernel\_size=3, activation='relu', input\_shape=input\_shape))  model.add(MaxPooling1D(pool\_size=2))  model.add(Flatten())  model.add(Dense(128, activation='relu'))  model.add(Dense(10, activation='softmax'))  model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy']) |

**Comparative analysis of different models trained on different features:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Features** | **Random Forest** | **KNN** | **LSTM** | **CNN** | **CNN+LSTM** |
| **Mel-Spectrogram** | **88.0%** | **54.46%** | **95.3%** | **99.2%** | **99.1%** |
| **Spectrogram** | **76.71%** | **96.58%** | **96.3%** | **99.4%** | **98.6%** |
| **Mel-spectrogram + MFCC** | **96.86%** | **95.05%** | **-** | **-** | **-** |
| **MFCC** | **88.0%** | **54.46%** | **98.20%** | **99.15%** | **-** |
| **Chroma** | **38.38%** | **34.61%** | **66.02%** | **72.83%** | **-** |
| **Chroma + MFCC** | **96.86%** | **95.05%** | **74.85%** | **87.63** | **-** |