ACQUIRING OF AUDIO-DATA SET FROM DRIVE:

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

EXTRACTION OF AUDIO FILES FROM ZIP FOLDER:

In [ ]:
```
import zipfile
import os
```

In [ ]:
```
zip_file_path = '/content/drive/MyDrive/Audio_files/Crema.zip'
extract_folder_path = '/content/drive/MyDrive/Audio_files/Crema'
```

In [ ]:
```
os.makedirs(extract_folder_path, exist_ok=True)
```

In [ ]:
```
pip install tqdm
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (4.67.1)
```

In [ ]:
```
from tqdm import tqdm
```

In [ ]:
```
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    files = zip_ref.namelist()

    # Use tqdm for a progress bar
    for file in tqdm(files, desc="Extracting Files", unit="file"):
        zip_ref.extract(file, extract_folder_path)
```

```
Extracting Files: 100%|██████████| 7442/7442 [31:33<00:00,  3.93file/s]
```

In [ ]:
```
 extracted_files = os.listdir(extract_folder_path)
```

In [ ]:
```
print(extracted_files[:10])
```

```
['1008_DFA_HAP_XX.wav', '1008_DFA_NEU_XX.wav', '1008_DFA_FEA_XX.wav', '1008_DFA_SAD_XX.wav', '1008_IEO_ANG_LO.wav', '1008_IEO_ANG_HI.wav', '1008_IEO
```

In [ ]:
```
import tensorflow as tf
import os
```

In [ ]:
```
audio_folder_path = '/content/drive/MyDrive/Audio_files/Crema'
```

In [ ]:
```
audio_files = os.listdir(audio_folder_path)
```

In [ ]:
```
def load_audio(filename):
    audio_binary = tf.io.read_file(filename)
    audio, sample_rate = tf.audio.decode_wav(audio_binary)
    return audio, sample_rate
```

In [ ]:
```
audio_file_path = os.path.join(audio_folder_path, audio_files[0])
audio, sample_rate = load_audio(audio_file_path)
```

In [ ]:
```
print(f"Audio shape: {audio.shape}")
print(f"Sample rate: {sample_rate}")
```

```
Audio shape: (51251, 1)
Sample rate: 16000
```

EXTRACTION OF FEATURES USING MEL-SPECTROGRAM WITH TENSORFLOW:

In [ ]:
```
import os
import librosa
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

```
from tensorflow.keras import layers, models
from tqdm import tqdm
```

In [ ]:
```
audio_folder_path = '/content/drive/MyDrive/Audio_files/Crema'
```

In [ ]:
```
def extract_mel_features(audio, sample_rate=16000, n_mels=64):  # Reduced n_mels for faster computation
    spectrogram = tf.signal.stft(audio, frame_length=1024, frame_step=512)
    spectrogram = tf.abs(spectrogram)
    mel_filter = tf.signal.linear_to_mel_weight_matrix(num_mel_bins=n_mels, num_spectrogram_bins=spectrogram.sh
    mel_spectrogram = tf.matmul(spectrogram, mel_filter)
    mel_spectrogram_db = tf.math.log(mel_spectrogram + 1e-6)
    return mel_spectrogram_db.numpy()
```

In [ ]:
```
features = []
labels = []
```

In [ ]:
```
audio_files = os.listdir(audio_folder_path)
```

In [ ]:
```
for file in tqdm(audio_files):
    try:

        audio_path = os.path.join(audio_folder_path, file)
        audio, sr = librosa.load(audio_path, sr=16000)


        mel_features = extract_mel_features(audio)
        mel_features_flat = mel_features.flatten()
        features.append(mel_features_flat)


        labels.append(1 if "positive" in file else 0)
    except Exception as e:
        print(f"Error processing file {file}: {e}")
```

```
100%|██████████| 7442/7442 [06:58<00:00, 17.79it/s]
```

TRAINING,TESTING AND SPLITTING OF DATASET:

In [ ]:
```
import tensorflow as tf
FIXED_SIZE = 128 * 64
```

In [ ]:
```
def pad_or_truncate(features, target_size=FIXED_SIZE):
    if len(features) > target_size:
        return features[:target_size]
    else:
        return np.pad(features, (0, target_size - len(features)), mode='constant')
```

In [ ]:
```
features_padded = [pad_or_truncate(feature.flatten(), FIXED_SIZE) for feature in features]
```

In [ ]:
```
X = np.array(features_padded)
y = np.array(labels)
```

In [ ]:
```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

IMPLEMENTATION OF CONVOLUTION NEURAL NETWORKS-ML MODEL:

In [ ]:
```
FIXED_SHAPE = (128, 64)
model = models.Sequential([
    layers.InputLayer(shape=(FIXED_SHAPE[0], FIXED_SHAPE[1], 1)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
X_train = X_train.reshape(-1, FIXED_SHAPE[0], FIXED_SHAPE[1], 1)
X_test = X_test.reshape(-1, FIXED_SHAPE[0], FIXED_SHAPE[1], 1)

model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))
```

```
Epoch 1/5
[1m94/94 [0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m45s [0m 429ms/step - accuracy: 0.9890 - loss: 0.0335 - val_accuracy: 1.0000 - val_lo
Epoch 2/5
[1m94/94 [0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m43s [0m 447ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - va
Epoch 3/5
[1m94/94 [0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m90s [0m 536ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - va
Epoch 4/5
[1m94/94 [0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m44s [0m 464ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - va
Epoch 5/5
[1m94/94 [0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m81s [0m 451ms/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - va
```

Out [39]: <keras.src.callbacks.history.History at 0x7eed5edb1a20>

In [ ]:
```
loss, accuracy = model.evaluate(X_test, y_test)
```

```
[1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m4s [0m 93ms/step - accuracy: 1.0000 - loss: 0.0000e+00
```

In [ ]:
```
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

```
Test Loss: 0.0000
Test Accuracy: 1.0000
```

In [ ]:
```
y_pred = model.predict(X_test)
```

```
[1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m4s [0m 76ms/step
```

In [ ]:
```
y_pred_binary = (y_pred > 0.5).astype(int)
```

In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_binary))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1489

    accuracy                           1.00      1489
   macro avg       1.00      1.00      1.00      1489
weighted avg       1.00      1.00      1.00      1489
```

In [ ]:
```
model.save('Audio_recognition.keras')
```

In [ ]:
```
from tensorflow.keras.models import load_model
from tensorflow.keras.optimizers import RMSprop
```

In [ ]:
```
model = load_model('Audio_recognition.keras', compile=False)
model.compile(optimizer=RMSprop(), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 126, 62, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 63, 31, 32)	0
flatten_1 (Flatten)	(None, 62496)	0
dense_2 (Dense)	(None, 128)	7,999,616
dense_3 (Dense)	(None, 1)	129

Total params: 8,000,065 (30.52 MB)

Trainable params: 8,000,065 (30.52 MB)

Non-trainable params: 0 (0.00 B)

In [ ]: