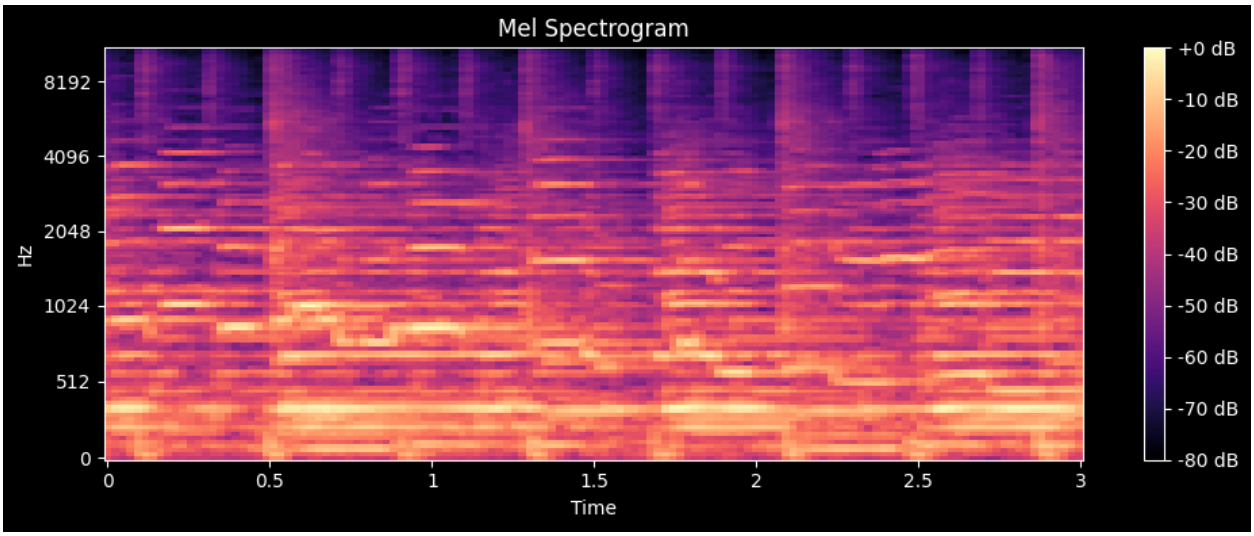
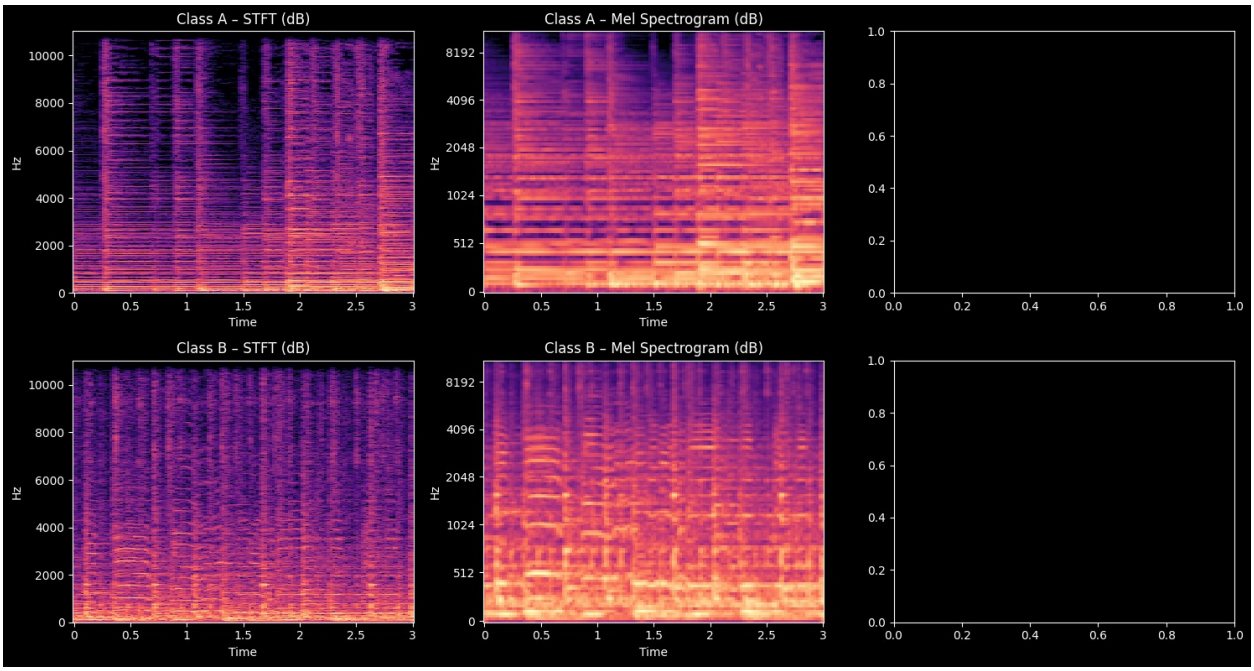
```
#TASK 1: GENERATE DEMO SPECTROGRAM
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

audio_path = (
    PROJECT_ROOT
    / "Data"
    / "IRMAS_Mono_files"
    / "train"
    / "cla"
    / "[cla][cla]0159__1.wav"
)
y, sr = librosa.load(audio_path, sr=22050, mono=True)

# Generate Mel Spectrogram
mel_spec = librosa.feature.melspectrogram(
    y=y,
    sr=sr,
    n_mels=128,
    fmax=sr // 2
)

mel_db = librosa.power_to_db(mel_spec, ref=np.max)

plt.figure(figsize=(10, 4))
librosa.display.specshow(
    mel_db,
    sr=sr,
    x_axis="time",
    y_axis="mel"
)
plt.colorbar(format="%+2.0f dB")
plt.title("Mel Spectrogram")
plt.tight_layout()
plt.show()
```




```
#TASK 2: COMPARE SPECTROGRAMS OF DIFFERENT CLASSES
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

audio_a = (
    PROJECT_ROOT / "Data" / "IRMAS_Mono_files" / "train" / "pia"
    / "[pia][pop_roc]1334__3.wav"
)

audio_b = (
    PROJECT_ROOT / "Data" / "IRMAS_Mono_files" / "train" / "cla"
    / "[cla][jaz_blu]0191__2.wav"
)


sr = 22050

y_a, _ = librosa.load(audio_a, sr=sr, mono=True)
y_b, _ = librosa.load(audio_b, sr=sr, mono=True)


def extract_features(y, sr):
    # STFT
    stft = np.abs(librosa.stft(y, n_fft=2048, hop_length=512))
    stft_db = librosa.amplitude_to_db(stft, ref=np.max)

    # Mel Spectrogram
    mel = librosa.feature.melspectrogram(
        y=y, sr=sr, n_mels=128
    )
    mel_db = librosa.power_to_db(mel, ref=np.max)
```

```
    return stft_db, mel_db

# Extract features
stft_a, mel_a = extract_features(y_a, sr)
stft_b, mel_b = extract_features(y_b, sr)

fig, axes = plt.subplots(2, 3, figsize=(15, 8))

features = [
    (stft_a, stft_b, "STFT (dB)", "linear"),
    (mel_a, mel_b, "Mel Spectrogram (dB)", "mel"),
]

for col, (fa, fb, title, y_axis) in enumerate(features):
    librosa.display.specshow(
        fa, sr=sr, x_axis="time", y_axis=y_axis, ax=axes[0, col]
    )
    axes[0, col].set_title(f"Class A — {title}")

    librosa.display.specshow(
        fb, sr=sr, x_axis="time", y_axis=y_axis, ax=axes[1, col]
    )
    axes[1, col].set_title(f"Class B — {title}")

plt.tight_layout()
plt.show()
print("Instruments with strong harmonic structures display distinct,
uniformly spaced bands throughout the spectrum because the STFT
spectrogram retains the entire linear-frequency content of the audio,
resulting in rich, finely resolved horizontal harmonic lines. The Mel-
spectrogram uses perceptual frequency scaling, which emphasizes low-
frequency energy and compresses high-frequency detail. As a result,
the same harmonics appear thicker, smoother, and more clustered,
mimicking how human hearing groups frequencies.")
```



Instruments with strong harmonic structures display distinct, uniformly spaced bands throughout the spectrum because the STFT spectrogram retains the entire linear-frequency content of the audio, resulting in rich, finely resolved horizontal harmonic lines. The Mel-spectrogram uses perceptual frequency scaling, which emphasizes low-frequency energy and compresses high-frequency detail. As a result, the same harmonics appear thicker, smoother, and more clustered, mimicking how human hearing groups frequencies.

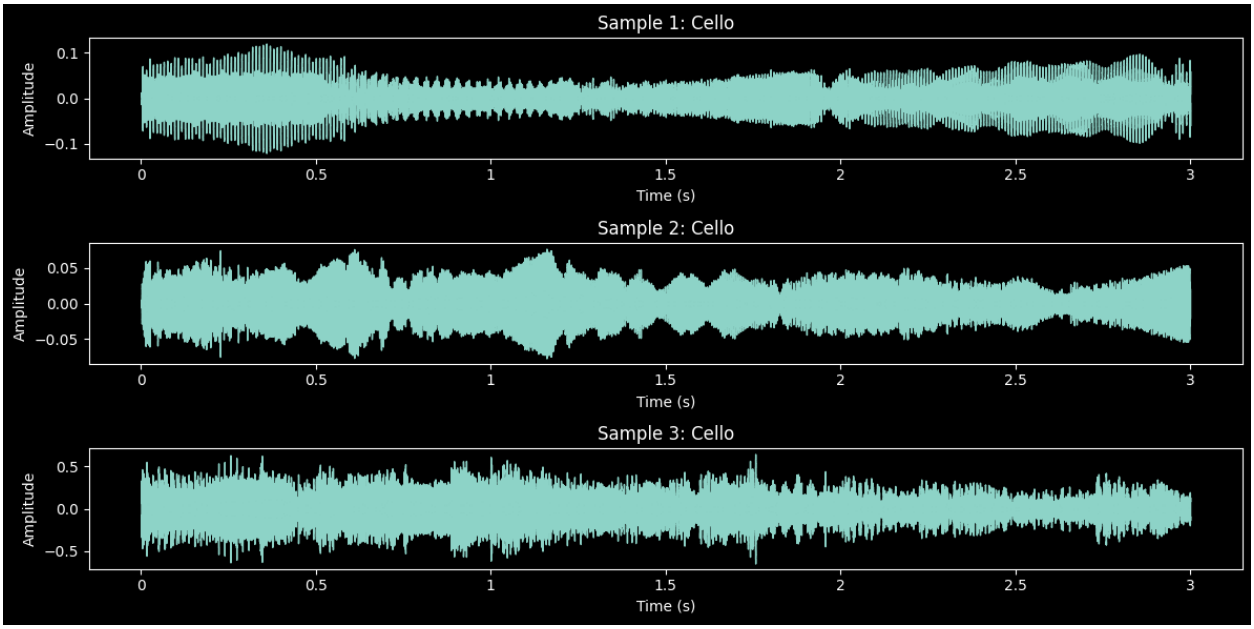
```
#TASK 3:

import os
import random
import librosa
import librosa.display
import matplotlib.pyplot as plt

# ---------------------------------------------------
# CONFIG
# ---------------------------------------------------
DATASET_ROOT = (
    PROJECT_ROOT
    / "Data"
    / "IRMAS-TrainingData"
    / "IRMAS-TrainingData"
    / "cel"
)
SR = 22050
NUM_SAMPLES = 3
```

```
# IRMAS instrument abbreviations
INSTRUMENT_MAP = {
    "cel": "Cello",
    "cla": "Clarinet",
    "flu": "Flute",
    "gac": "Acoustic Guitar",
    "gel": "Electric Guitar",
    "org": "Organ",
    "pia": "Piano",
    "sax": "Saxophone",
    "tru": "Trumpet",
    "vio": "Violin",
    "voi": "Human Voice"
}

# ----------------------------------------------------
# LOAD DATASET FILE LIST
# ----------------------------------------------------
audio_files = []

for root, _, files in os.walk(DATASET_ROOT):
    for f in files:
        if f.endswith(".wav"):
            audio_files.append(os.path.join(root, f))

print(f"Total audio files found: {len(audio_files)}")

# ----------------------------------------------------
# SELECT RANDOM SAMPLES
# ----------------------------------------------------
selected_files = random.sample(audio_files, NUM_SAMPLES)

# ----------------------------------------------------
# DISPLAY INSTRUMENT MAP
# ----------------------------------------------------
print("\n♪ Instrument Map (IRMAS)")
print("-" * 30)
for k, v in INSTRUMENT_MAP.items():
    print(f"{k.upper():<8} → {v}")

# ----------------------------------------------------
# DISPLAY SAMPLE AUDIOS
# ----------------------------------------------------
plt.figure(figsize=(12, 6))

for idx, file_path in enumerate(selected_files):
    filename = os.path.basename(file_path).lower()

    # Detect instrument label from filename
    label = "Unknown"
```

```
    for key in INSTRUMENT_MAP:
        if key in filename:
            label = INSTRUMENT_MAP[key]
            break

    # Load audio
    y, sr = librosa.load(file_path, sr=SR, mono=True)

    # Plot waveform
    plt.subplot(NUM_SAMPLES, 1, idx + 1)
    librosa.display.waveshow(y, sr=sr)
    plt.title(f"Sample {idx+1}: {label}")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")

    print(f"\n Sample {idx+1}")
    print(f"File  : {file_path}")
    print(f"Label : {label}")

plt.tight_layout()
plt.show()
```

```
Total audio files found: 388

♪ Instrument Map (IRMAS)
------------------------------
CEL      → Cello
CLAR     → Clarinet
FLUTE    → Flute
GAC      → Acoustic Guitar
GEL      → Electric Guitar
ORGAN    → Organ
PIANO    → Piano
SAX      → Saxophone
TRUMPET  → Trumpet
VIOLIN   → Violin
VOICE    → Human Voice

 Sample 1
File  : C:\Games\Instrunet\Data\IRMAS-TrainingData\IRMAS-TrainingData\
cel\115__[cel][nod][cla]0055__2.wav
Label : Cello

 Sample 2
File  : C:\Games\Instrunet\Data\IRMAS-TrainingData\IRMAS-TrainingData\
cel\115__[cel][nod][cla]0055__1.wav
Label : Cello

 Sample 3
File  : C:\Games\Instrunet\Data\IRMAS-TrainingData\IRMAS-TrainingData\
```

```
cel\[cel][pop_roc]0103__1.wav
Label : Cello
```



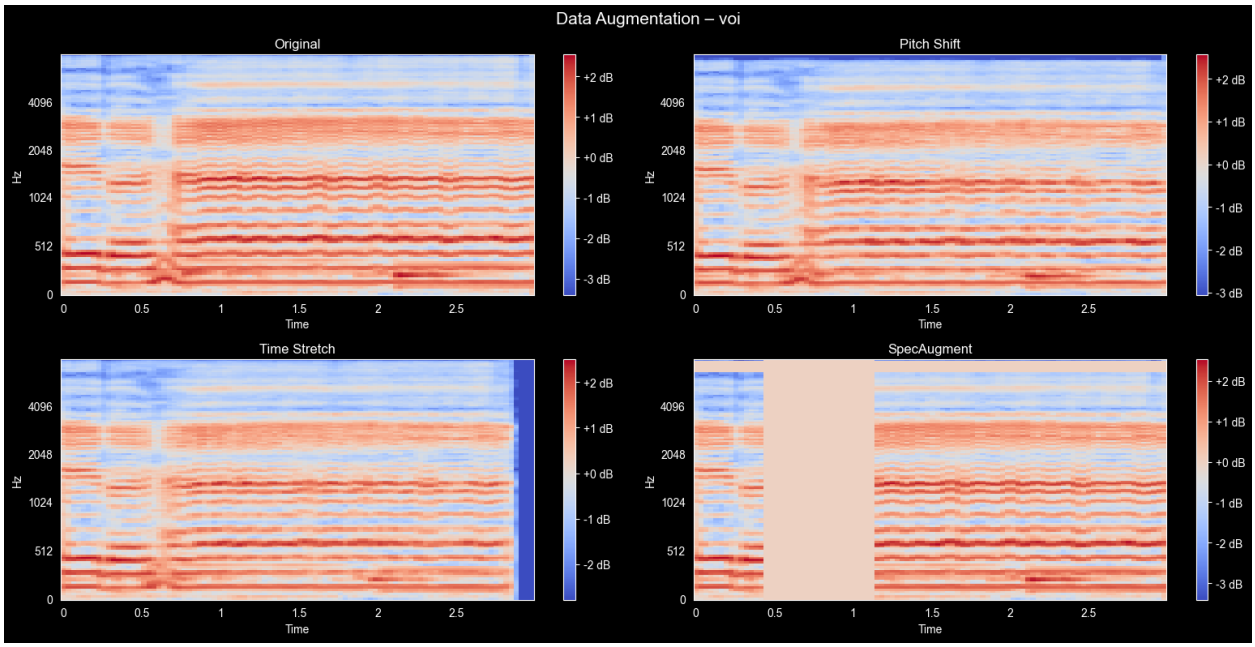

```python
#TASK 4: DATA AUGMENTATION
import os
import random
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt
from tqdm import tqdm

# ==================================================
# PATHS
# ==================================================
IRMAS_MONO_ROOT = PROJECT_ROOT / "Data" / "IRMAS_Mono_files"
OUTPUT_ROOT = PROJECT_ROOT / "Data" / "Processed" / "IRMAS_Mono"

# ==================================================
# CONSTANTS
# ==================================================
TARGET_SR = 16000
FIXED_DURATION = 3.0  # seconds
N_MELS = 128

N_FFT = 2048
HOP_LENGTH = 512
WIN_LENGTH = 2048
WINDOW = "hann"
```

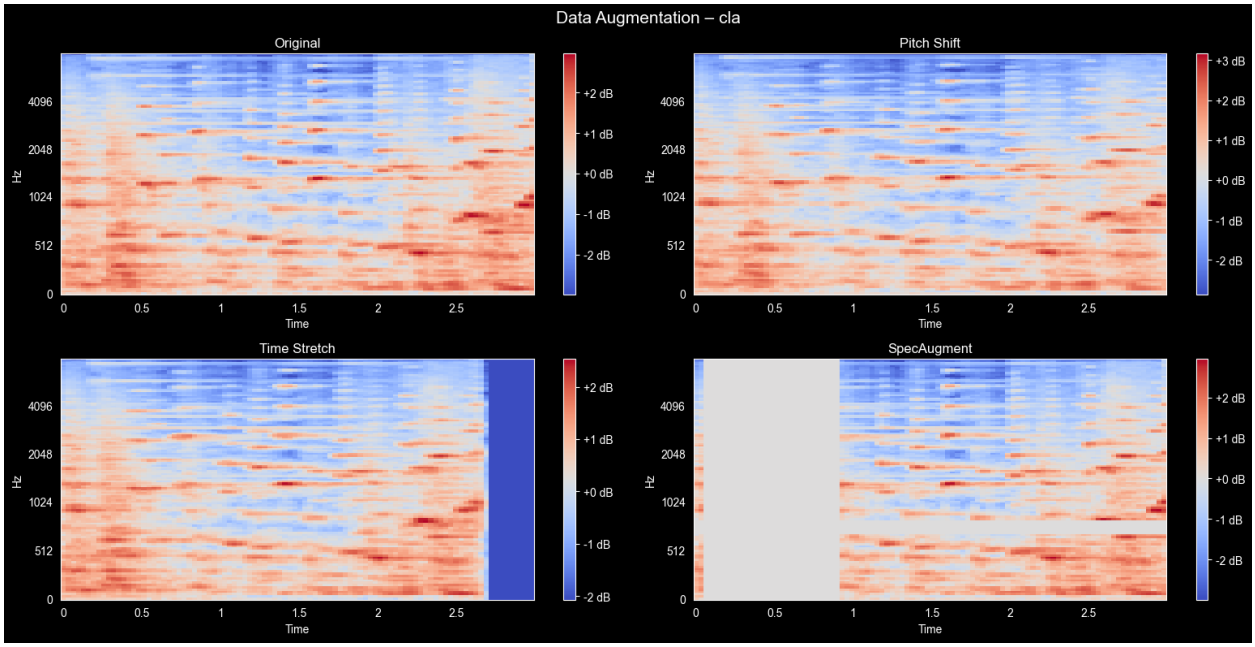
```python
# ================================================
# AUDIO PREPROCESSING FUNCTIONS
# ================================================
def load_audio(path):
    try:
        audio, sr = librosa.load(path, sr=None, mono=False)
        return audio, sr
    except Exception as e:
        print(f"[CORRUPTED] {os.path.basename(path)} | {e}")
        return None, None


def stereo_to_mono(audio):
    if audio.ndim == 1:
        return audio
    return np.mean(audio, axis=0)


def resample_audio(audio, orig_sr, target_sr=TARGET_SR):
    if orig_sr != target_sr:
        audio = librosa.resample(audio, orig_sr=orig_sr,
target_sr=target_sr)
    return audio


def peak_normalize(audio):
    peak = np.max(np.abs(audio))
    return audio / peak if peak > 0 else audio


def trim_silence(audio, thresh=0.02):
    idx = np.where(np.abs(audio) > thresh)[0]
    if len(idx) == 0:
        return audio
    return audio[idx[0]: idx[-1]]


def fix_duration(audio, sr=TARGET_SR, duration=FIXED_DURATION):
    target_len = int(sr * duration)
    if len(audio) > target_len:
        return audio[:target_len]
    else:
        return np.pad(audio, (0, target_len - len(audio)),
mode="constant")


# ================================================
# LOG-MEL GENERATION
# ================================================
def generate_log_mel(audio, sr=TARGET_SR):
```

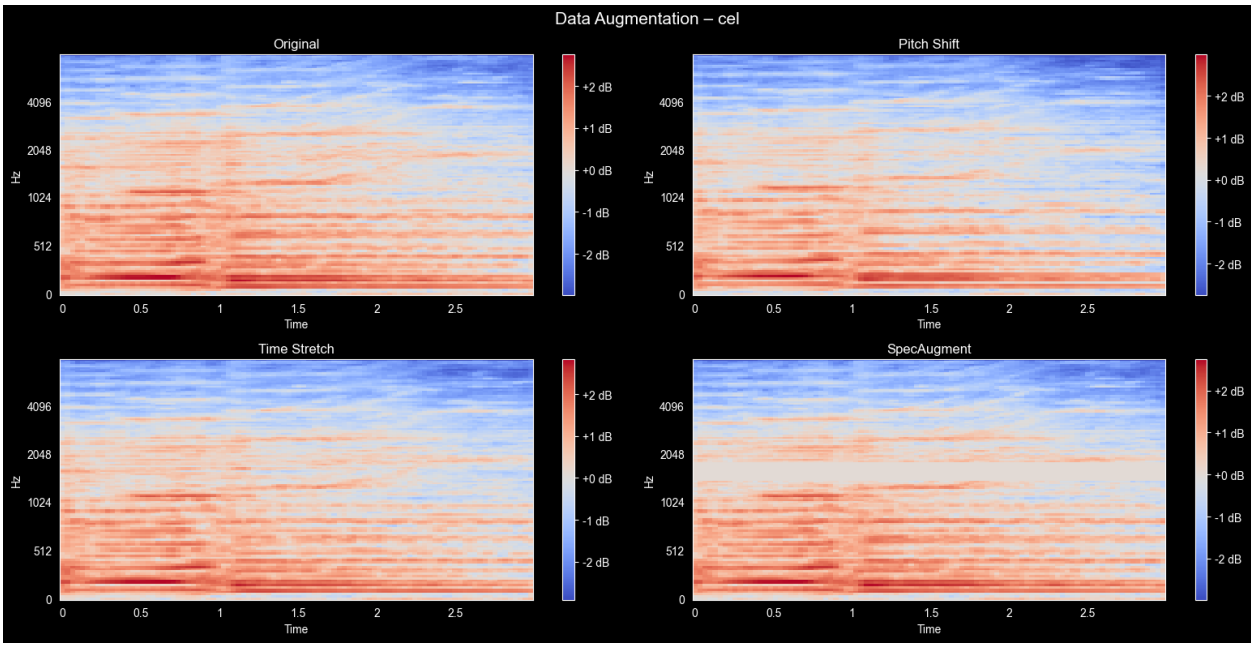
```
    mel = librosa.feature.melspectrogram(
        y=audio,
        sr=sr,
        n_fft=N_FFT,
        hop_length=HOP_LENGTH,
        win_length=WIN_LENGTH,
        window=WINDOW,
        n_mels=N_MELS,
        power=2.0
    )

    mel_db = librosa.power_to_db(mel, ref=np.max)

    # Per-sample normalization
    mel_db = (mel_db - mel_db.mean()) / (mel_db.std() + 1e-8)

    return mel_db


# ================================================
# FRAME CONSISTENCY (DURATION LOCKED)
# ================================================
def fix_mel_frames(mel, sr=TARGET_SR, duration=FIXED_DURATION,
hop_length=HOP_LENGTH):
    target_frames = int(np.ceil((sr * duration) / hop_length))
    current_frames = mel.shape[1]

    if current_frames < target_frames:
        pad_width = target_frames - current_frames
        mel = np.pad(mel, ((0, 0), (0, pad_width)), mode="constant")
    else:
        mel = mel[:, :target_frames]

    return mel


# ================================================
# DATA AUGMENTATION FUNCTIONS
# ================================================
def augment_pitch_shift(audio, sr=TARGET_SR, max_steps=2):
    steps = np.random.uniform(-max_steps, max_steps)
    return librosa.effects.pitch_shift(audio, sr=sr, n_steps=steps)


def augment_time_stretch(audio, rate_range=(0.8, 1.25)):
    rate = np.random.uniform(*rate_range)
    stretched = librosa.effects.time_stretch(audio, rate=rate)

    # Fix length back to original
    if len(stretched) > len(audio):
```

```
        stretched = stretched[:len(audio)]
    else:
        stretched = np.pad(stretched, (0, len(audio) -
len(stretched)))

    return stretched


def spec_augment(mel, freq_mask=20, time_mask=30):
    mel_aug = mel.copy()

    # Frequency masking
    f = np.random.randint(0, freq_mask)
    f0 = np.random.randint(0, mel.shape[0] - f)
    mel_aug[f0:f0 + f, :] = 0

    # Time masking
    t = np.random.randint(0, time_mask)
    t0 = np.random.randint(0, mel.shape[1] - t)
    mel_aug[:, t0:t0 + t] = 0

    return mel_aug


# ================================================
# AUGMENTATION VISUALIZATION
# ================================================
def visualize_augmentations(audio_path, class_name):
    # -------- LOAD & BASE PROCESS --------
    audio, sr = load_audio(audio_path)
    audio = stereo_to_mono(audio)
    audio = resample_audio(audio, sr)
    audio = peak_normalize(audio)
    audio = trim_silence(audio)
    audio = fix_duration(audio)

    # -------- ORIGINAL MEL --------
    mel_orig = fix_mel_frames(generate_log_mel(audio))

    # -------- AUGMENTATIONS --------
    audio_pitch = augment_pitch_shift(audio)
    audio_time = augment_time_stretch(audio)

    mel_pitch = fix_mel_frames(generate_log_mel(audio_pitch))
    mel_time = fix_mel_frames(generate_log_mel(audio_time))
    mel_specaug = fix_mel_frames(spec_augment(mel_orig))

    # -------- PLOTTING --------
    fig, axes = plt.subplots(2, 2, figsize=(16, 8))
    fig.suptitle(f"Data Augmentation – {class_name}", fontsize=15)
```

```
    plots = [
        ("Original", mel_orig),
        ("Pitch Shift", mel_pitch),
        ("Time Stretch", mel_time),
        ("SpecAugment", mel_specaug),
    ]

    for ax, (title, mel) in zip(axes.flat, plots):
        img = librosa.display.specshow(
            mel,
            sr=TARGET_SR,
            hop_length=HOP_LENGTH,
            x_axis="time",
            y_axis="mel",
            ax=ax
        )
        ax.set_title(title)
        fig.colorbar(img, ax=ax, format="%+2.0f dB")

    plt.tight_layout()
    plt.show()


# ================================================
# DEMO: VISUALIZE AUGMENTATIONS
# ================================================
random.seed(42)

split = "train"
split_dir = os.path.join(IRMAS_MONO_ROOT, split)

classes = random.sample(os.listdir(split_dir), 3)

for class_name in classes:
    class_dir = os.path.join(split_dir, class_name)
    wav_files = [f for f in os.listdir(class_dir) if
f.endswith(".wav")]

    if not wav_files:
        continue

    audio_file = random.choice(wav_files)
    audio_path = os.path.join(class_dir, audio_file)

    visualize_augmentations(audio_path, class_name)
```


Data Augmentation – voi
Original
Pitch Shift
Time Stretch
SpecAugment
Hz
Time
4096
2048
1024
512
0
0.5
1
1.5
2
2.5
+2 dB
+1 dB
+0 dB
-1 dB
-2 dB
-3 dB


Data Augmentation – cla
Original
Pitch Shift
Time Stretch
SpecAugment
Hz
Time
4096
2048
1024
512
0
0.5
1
1.5
2
2.5
+3 dB
+2 dB
+1 dB
+0 dB
-1 dB
-2 dB




```
## TASK 5 and 6: TRAINING and HYPERPAREMETER TUNING
import os
import json
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import (
    EarlyStopping,
    ReduceLROnPlateau,
    ModelCheckpoint
)
from tensorflow.keras import backend as K
from sklearn.metrics import classification_report, confusion_matrix
from pathlib import Path

# ======================================
# PROJECT ROOT (works in .py & Jupyter)
# ======================================
try:
    PROJECT_ROOT = Path(__file__).resolve().parent.parent
except NameError:
    # Jupyter notebook fallback
    PROJECT_ROOT = Path.cwd().parent

PROJECT_ROOT = PROJECT_ROOT.resolve()

print("PROJECT_ROOT:", PROJECT_ROOT)
```

```
# ================================
# Global Configuration
# ================================
DATA_ROOT = os.path.join(
    PROJECT_ROOT,
    "Data",
    "Processed",
    "IRMAS_Mono"
)

INPUT_SHAPE = (128, 126, 1)
NUM_CLASSES = 11

BATCH_SIZE = 64
EPOCHS = 40
SEED = 42

tf.random.set_seed(SEED)
np.random.seed(SEED)
# ================================
# Load label mapping
# ================================
LABEL_MAP_PATH = os.path.join(
    PROJECT_ROOT,
    "Data",
    "Split",
    "label_map.json"
)

with open(LABEL_MAP_PATH, "r") as f:
    class_to_id = json.load(f)

id_to_class = {v: k for k, v in class_to_id.items()}

print("Classes:", class_to_id)
# ================================
# Data loading (single-label)
# ================================
def load_split(split):
    X, y = [], []
    split_dir = os.path.join(DATA_ROOT, split)

    for cls in os.listdir(split_dir):
        cls_id = class_to_id[cls]
        cls_dir = os.path.join(split_dir, cls)

        for file in os.listdir(cls_dir):
            if file.endswith(".npy"):
                mel = np.load(os.path.join(cls_dir, file))
                mel = mel[..., np.newaxis]
```

```
                label = tf.keras.utils.to_categorical(cls_id,
NUM_CLASSES)

                X.append(mel)
                y.append(label)

    return np.array(X), np.array(y)
# ==============================
# Load splits
# ==============================
X_train, y_train = load_split("train")
X_val, y_val     = load_split("val")
X_test, y_test   = load_split("test")

print("Train:", X_train.shape, y_train.shape)
print("Val  :", X_val.shape, y_val.shape)
print("Test :", X_test.shape, y_test.shape)
# ==============================
# CNN Architecture
# ==============================
def build_instrunet_irmas():
    inputs = layers.Input(shape=INPUT_SHAPE)

    def conv_block(x, filters):
        x = layers.Conv2D(
            filters,
            kernel_size=(5, 5),     #  CHANGED FROM (3,3) TO (5,5)
            padding="same"
        )(x)
        x = layers.BatchNormalization()(x)
        x = layers.Activation("relu")(x)
        x = layers.MaxPooling2D((2, 2))(x)
        return x

    x = conv_block(inputs, 32)
    x = conv_block(x, 64)
    x = conv_block(x, 128)
    x = conv_block(x, 256)

    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(256, activation="relu")(x)
    x = layers.Dropout(0.4)(x)

    outputs = layers.Dense(NUM_CLASSES, activation="softmax")(x)

    return models.Model(inputs, outputs)
# ==============================
# Compile
# ==============================
```

```
K.clear_session()

model = build_instrunet_irmas()

model.compile(
    optimizer=tf.keras.optimizers.SGD(
        learning_rate=1e-2,
        momentum=0.9,
        nesterov=True
    ),
    loss=tf.keras.losses.CategoricalCrossentropy(label_smoothing=0.1),
    metrics=[tf.keras.metrics.CategoricalAccuracy(name="accuracy")]
)

model.summary()
# ================================
# Callbacks
# ================================
callbacks = [
    EarlyStopping(
        monitor="val_loss",
        patience=8,
        restore_best_weights=True
    ),
    ReduceLROnPlateau(
        monitor="val_loss",
        factor=0.5,
        patience=4,
        min_lr=1e-6,
        verbose=1
    ),
    ModelCheckpoint(
        filepath="../Model/instrunet_irmas_sgd_finetuned.keras",
        monitor="val_loss",
        save_best_only=True,
        verbose=1
    )
]
# ================================
# Training
# ================================
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    callbacks=callbacks,
    verbose=1
)
# ================================
```

```
# Training curves
# ================================
plt.figure(figsize=(6,4))
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Val Accuracy")
plt.legend()
plt.title("Accuracy")
plt.show()

plt.figure(figsize=(6,4))
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.legend()
plt.title("Loss")
plt.show()
# ================================
# Evaluation
# ================================
y_prob = model.predict(X_test)
y_pred = np.argmax(y_prob, axis=1)
y_true = np.argmax(y_test, axis=1)

print("\nClassification Report:")
print(classification_report(
    y_true,
    y_pred,
    target_names=[id_to_class[i] for i in range(NUM_CLASSES)]
))
# ================================
# Confusion Matrix
# ================================
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8,6))
plt.imshow(cm, cmap="Blues")
plt.colorbar()
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()

MODEL_OUT = os.path.join(
    PROJECT_ROOT,
    "Model",
    "instrunet_irmas_sgd_finetuned.keras"
)

PROJECT_ROOT: C:\Games\Instrunet
Classes: {'cel': 0, 'cla': 1, 'flu': 2, 'gac': 3, 'gel': 4, 'org': 5,
'pia': 6, 'sax': 7, 'tru': 8, 'vio': 9, 'voi': 10}
Train: (4693, 128, 126, 1) (4693, 11)
```

```
Val  : (1006, 128, 126, 1) (1006, 11)
Test : (1006, 128, 126, 1) (1006, 11)

Model: "functional"
```

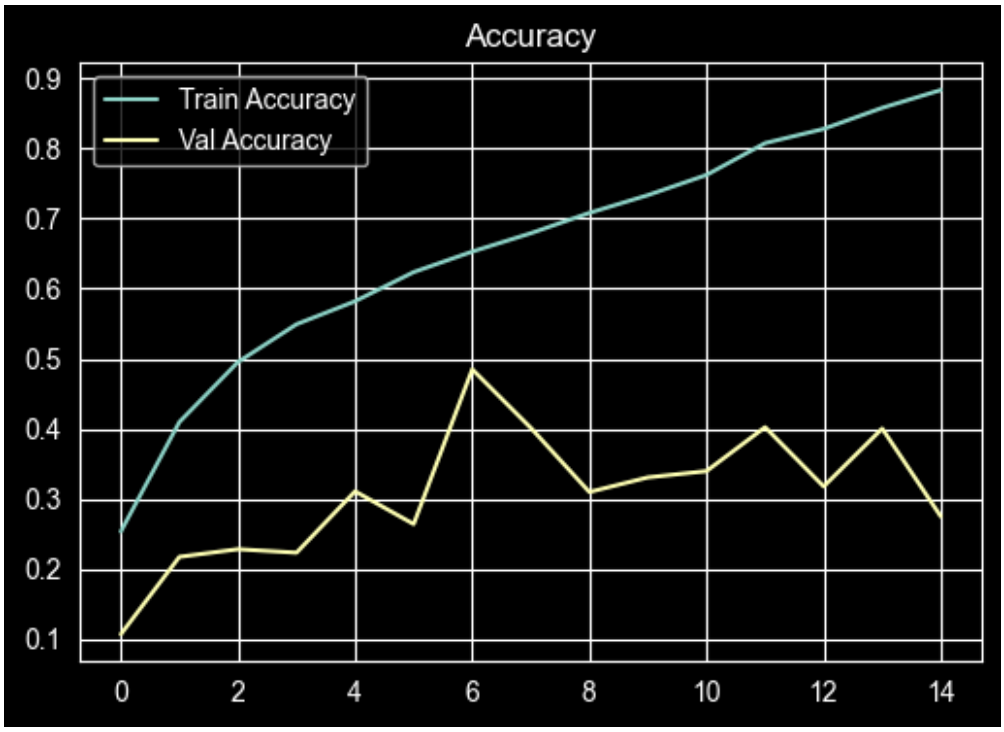
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 128, 126, 1)	0
conv2d (Conv2D)	(None, 128, 126, 32)	832
batch_normalization (BatchNormalization)	(None, 128, 126, 32)	128
activation (Activation)	(None, 128, 126, 32)	0
max_pooling2d (MaxPooling2D)	(None, 64, 63, 32)	0
conv2d_1 (Conv2D)	(None, 64, 63, 64)	51,264
batch_normalization_1 (BatchNormalization)	(None, 64, 63, 64)	256
activation_1 (Activation)	(None, 64, 63, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 32, 31, 64)	0

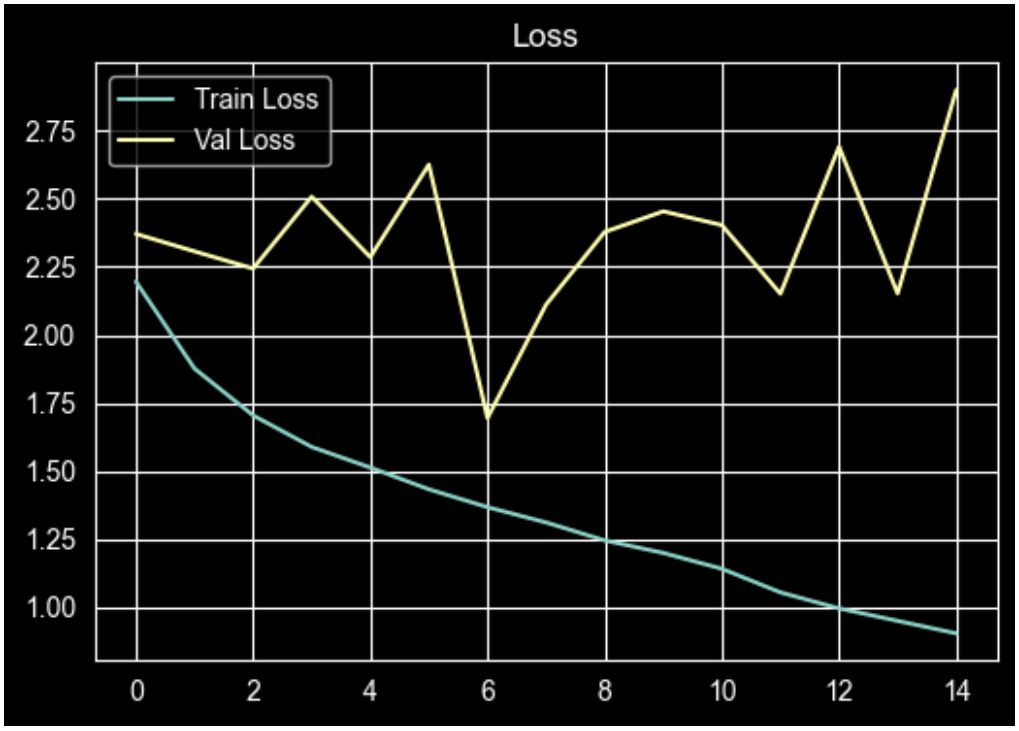
```
──────┤
| conv2d_2 (Conv2D)                | (None, 32, 31, 128)    |
204,928 |
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| batch_normalization_2            | (None, 32, 31, 128)    |
512 |
| (BatchNormalization)             |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| activation_2 (Activation)        | (None, 32, 31, 128)    |
0 |
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| max_pooling2d_2 (MaxPooling2D)   | (None, 16, 15, 128)    |
0 |
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| conv2d_3 (Conv2D)                | (None, 16, 15, 256)    |
819,456 |
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| batch_normalization_3            | (None, 16, 15, 256)    |
1,024 |
| (BatchNormalization)             |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| activation_3 (Activation)        | (None, 16, 15, 256)    |
0 |
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| max_pooling2d_3 (MaxPooling2D)   | (None, 8, 7, 256)      |
0 |
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| global_average_pooling2d         | (None, 256)            |
0 |
| (GlobalAveragePooling2D)         |                         |
|
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| dense (Dense)                    | (None, 256)            |
65,792 |
├──────────────────────────────────┼─────────────────────────┼──────
──────┤
| dropout (Dropout)                | (None, 256)            |
0 |
```

```
├──────────────────────────────────┼───────────────────────────┼──────────
──────┤
│ dense_1 (Dense)                  │ (None, 11)                │
2,827 │
└──────────────────────────────────┴───────────────────────────┴──────────
──────┘

 Total params: 1,147,019 (4.38 MB)

 Trainable params: 1,146,059 (4.37 MB)

 Non-trainable params: 960 (3.75 KB)

Epoch 1/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 861ms/step - accuracy: 0.2019 - loss:
2.3508
Epoch 1: val_loss improved from None to 2.37150, saving model to
../Model/instrunet_irmas_sgd_finetuned.keras

Epoch 1: finished saving model to
../Model/instrunet_irmas_sgd_finetuned.keras
74/74 ━━━━━━━━━━━━━━━━━━━━ 69s 908ms/step - accuracy: 0.2538 - loss:
2.1964 - val_accuracy: 0.1074 - val_loss: 2.3715 - learning_rate:
0.0100
Epoch 2/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 843ms/step - accuracy: 0.3804 - loss:
1.9311
Epoch 2: val_loss improved from 2.37150 to 2.30725, saving model
to ../Model/instrunet_irmas_sgd_finetuned.keras

Epoch 2: finished saving model to
../Model/instrunet_irmas_sgd_finetuned.keras
74/74 ━━━━━━━━━━━━━━━━━━━━ 66s 885ms/step - accuracy: 0.4100 - loss:
1.8770 - val_accuracy: 0.2177 - val_loss: 2.3072 - learning_rate:
0.0100
Epoch 3/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 854ms/step - accuracy: 0.4742 - loss:
1.7439
Epoch 3: val_loss improved from 2.30725 to 2.24432, saving model
to ../Model/instrunet_irmas_sgd_finetuned.keras

Epoch 3: finished saving model to
../Model/instrunet_irmas_sgd_finetuned.keras
74/74 ━━━━━━━━━━━━━━━━━━━━ 66s 895ms/step - accuracy: 0.4954 - loss:
1.7051 - val_accuracy: 0.2286 - val_loss: 2.2443 - learning_rate:
0.0100
Epoch 4/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 852ms/step - accuracy: 0.5397 - loss:
1.6170
Epoch 4: val_loss did not improve from 2.24432
```

```
74/74 ━━━━━━━━━━━━━━━━━━━━ 66s 895ms/step - accuracy: 0.5493 - loss:
1.5892 - val_accuracy: 0.2237 - val_loss: 2.5083 - learning_rate:
0.0100
Epoch 5/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 930ms/step - accuracy: 0.5713 - loss:
1.5386
Epoch 5: val_loss did not improve from 2.24432
74/74 ━━━━━━━━━━━━━━━━━━━━ 72s 974ms/step - accuracy: 0.5821 - loss:
1.5134 - val_accuracy: 0.3111 - val_loss: 2.2862 - learning_rate:
0.0100
Epoch 6/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 910ms/step - accuracy: 0.6073 - loss:
1.4475
Epoch 6: val_loss did not improve from 2.24432
74/74 ━━━━━━━━━━━━━━━━━━━━ 70s 952ms/step - accuracy: 0.6237 - loss:
1.4337 - val_accuracy: 0.2644 - val_loss: 2.6261 - learning_rate:
0.0100
Epoch 7/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 910ms/step - accuracy: 0.6423 - loss:
1.3854
Epoch 7: val_loss improved from 2.24432 to 1.69495, saving model
to ../Model/instrunet_irmas_sgd_finetuned.keras

Epoch 7: finished saving model to
../Model/instrunet_irmas_sgd_finetuned.keras
74/74 ━━━━━━━━━━━━━━━━━━━━ 71s 953ms/step - accuracy: 0.6529 - loss:
1.3680 - val_accuracy: 0.4851 - val_loss: 1.6950 - learning_rate:
0.0100
Epoch 8/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 908ms/step - accuracy: 0.6649 - loss:
1.3334
Epoch 8: val_loss did not improve from 1.69495
74/74 ━━━━━━━━━━━━━━━━━━━━ 70s 950ms/step - accuracy: 0.6791 - loss:
1.3119 - val_accuracy: 0.4016 - val_loss: 2.1118 - learning_rate:
0.0100
Epoch 9/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 910ms/step - accuracy: 0.6945 - loss:
1.2625
Epoch 9: val_loss did not improve from 1.69495
74/74 ━━━━━━━━━━━━━━━━━━━━ 71s 954ms/step - accuracy: 0.7081 - loss:
1.2459 - val_accuracy: 0.3101 - val_loss: 2.3786 - learning_rate:
0.0100
Epoch 10/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 908ms/step - accuracy: 0.7296 - loss:
1.2118
Epoch 10: val_loss did not improve from 1.69495
74/74 ━━━━━━━━━━━━━━━━━━━━ 70s 951ms/step - accuracy: 0.7334 - loss:
1.2000 - val_accuracy: 0.3310 - val_loss: 2.4543 - learning_rate:
0.0100
```

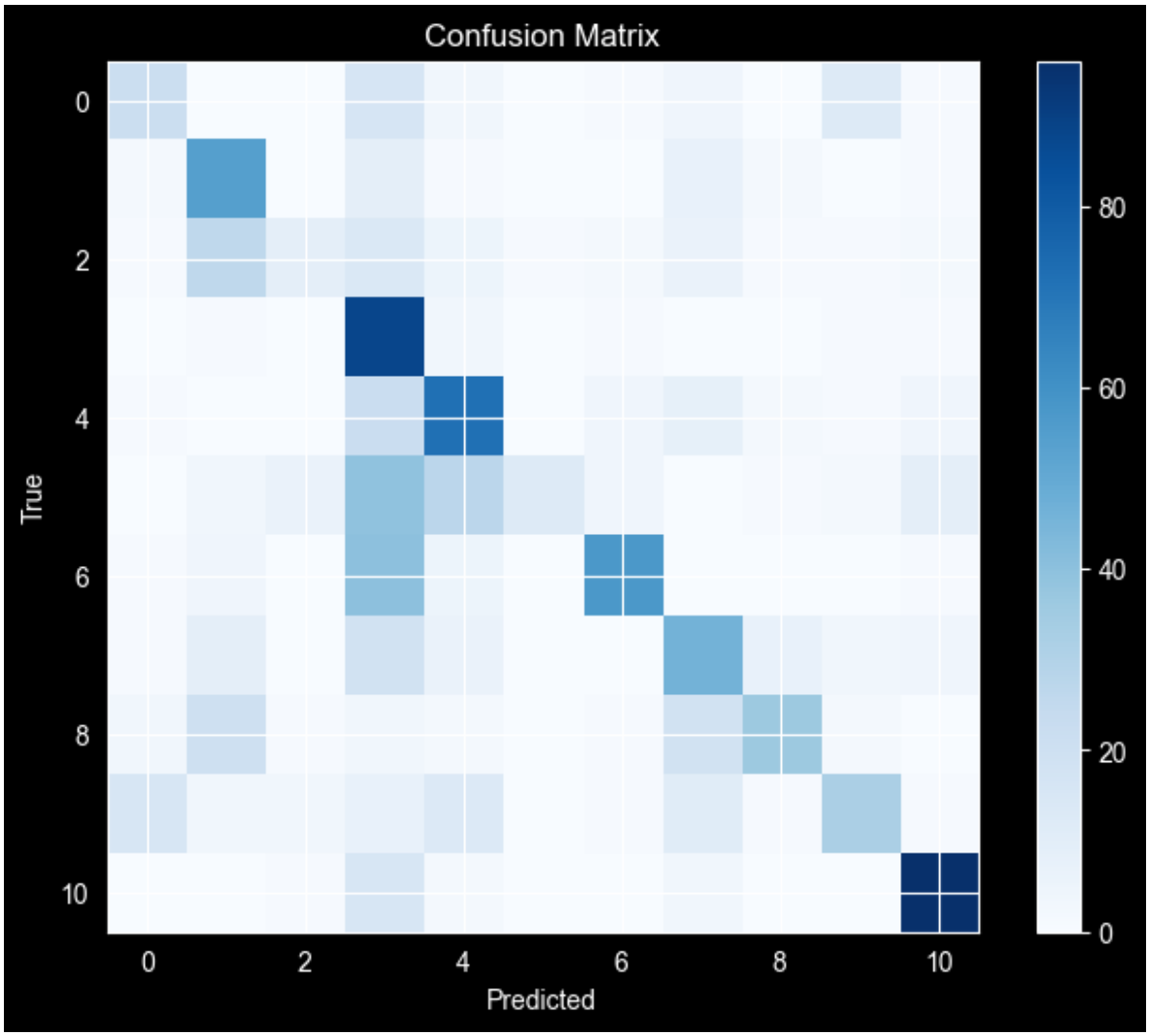
```
Epoch 11/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 920ms/step - accuracy: 0.7553 - loss:
1.1541
Epoch 11: ReduceLROnPlateau reducing learning rate to
0.004999999888241291.

Epoch 11: val_loss did not improve from 1.69495
74/74 ━━━━━━━━━━━━━━━━━━━━ 71s 962ms/step - accuracy: 0.7622 - loss:
1.1417 - val_accuracy: 0.3400 - val_loss: 2.4039 - learning_rate:
0.0100
Epoch 12/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 912ms/step - accuracy: 0.7920 - loss:
1.0931
Epoch 12: val_loss did not improve from 1.69495
74/74 ━━━━━━━━━━━━━━━━━━━━ 71s 957ms/step - accuracy: 0.8076 - loss:
1.0551 - val_accuracy: 0.4026 - val_loss: 2.1518 - learning_rate:
0.0050
Epoch 13/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 866ms/step - accuracy: 0.8118 - loss:
1.0242
Epoch 13: val_loss did not improve from 1.69495
74/74 ━━━━━━━━━━━━━━━━━━━━ 79s 914ms/step - accuracy: 0.8278 - loss:
0.9962 - val_accuracy: 0.3181 - val_loss: 2.6920 - learning_rate:
0.0050
Epoch 14/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 863ms/step - accuracy: 0.8442 - loss:
0.9697
Epoch 14: val_loss did not improve from 1.69495
74/74 ━━━━━━━━━━━━━━━━━━━━ 67s 912ms/step - accuracy: 0.8579 - loss:
0.9505 - val_accuracy: 0.4006 - val_loss: 2.1526 - learning_rate:
0.0050
Epoch 15/40
74/74 ━━━━━━━━━━━━━━━━━━━━ 0s 868ms/step - accuracy: 0.8759 - loss:
0.9240
Epoch 15: ReduceLROnPlateau reducing learning rate to
0.0024999999441206455.

Epoch 15: val_loss did not improve from 1.69495
74/74 ━━━━━━━━━━━━━━━━━━━━ 68s 915ms/step - accuracy: 0.8832 - loss:
0.9045 - val_accuracy: 0.2753 - val_loss: 2.9001 - learning_rate:
0.0050
```






```
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 115ms/step

Classification Report:
              precision    recall  f1-score   support

         cel       0.47      0.36      0.41        58
```

cla	0.45	0.71	0.55	76
flu	0.45	0.13	0.20	68
gac	0.32	0.93	0.48	95
gel	0.52	0.63	0.57	114
org	0.92	0.12	0.21	103
pia	0.80	0.53	0.64	108
sax	0.45	0.49	0.47	94
tru	0.72	0.42	0.53	86
vio	0.59	0.37	0.45	87
voi	0.80	0.82	0.81	117
accuracy			0.52	1006
macro avg	0.59	0.50	0.48	1006
weighted avg	0.61	0.52	0.50	1006



```
## TASK 7: TESTING
import os
import random
import numpy as np
import librosa
import tensorflow as tf
from tqdm import tqdm

from sklearn.metrics import (
    classification_report,
    f1_score,
    accuracy_score,
    hamming_loss
)


# =========================
# GLOBAL CONFIG
# =========================
from pathlib import Path

# ======================================
# PROJECT ROOT (works in .py & Jupyter)
# ======================================
try:
    PROJECT_ROOT = Path(__file__).resolve().parent.parent
except NameError:
    # Jupyter notebook fallback
    PROJECT_ROOT = Path.cwd().parent

PROJECT_ROOT = PROJECT_ROOT.resolve()

print("PROJECT_ROOT:", PROJECT_ROOT)

# ======================================
# PATHS (STREAMLIT SAFE)
# ======================================
TEST_AUDIO_ROOT = (
    PROJECT_ROOT
    / "Data"
    / "Testing"
    / "IRMAS-TestingData-Part1"
    / "Part1"
)

MODEL_PATH = PROJECT_ROOT / "Model" /
"instrunet_irmas_sgd_finetuned.keras"


TARGET_SR = 16000
WINDOW_SEC = 3.0
```

```
HOP_SEC = 1.5
N_MELS = 128
TARGET_FRAMES = 126
NUM_CLASSES = 11
EPS = 1e-8

RANDOM_SEED = 42
NUM_TEST_FILES = 500


# ==========================
# CLASS DEFINITIONS
# ==========================
CLASS_NAMES = [
    "cel", "cla", "flu", "gac", "gel",
    "org", "pia", "sax", "tru", "vio", "voi"
]

CLASS_THRESHOLDS = {
    "cel": 0.35,
    "cla": 0.40,
    "flu": 0.40,
    "gac": 0.25,
    "gel": 0.25,
    "org": 0.35,
    "pia": 0.20,
    "sax": 0.45,
    "tru": 0.40,
    "vio": 0.30,
    "voi": 0.50,
}

class_to_id = {c: i for i, c in enumerate(CLASS_NAMES)}
id_to_class = {i: c for c, i in class_to_id.items()}


# ==========================
# LOAD MODEL
# ==========================
model = tf.keras.models.load_model(MODEL_PATH)
model.summary()


# ==========================
# AUDIO PREPROCESSING
# ==========================
def stereo_to_mono(audio):
    if audio.ndim == 1:
        return audio
    return np.mean(audio, axis=0)
```

```
def peak_normalize(audio):
    peak = np.max(np.abs(audio))
    return audio / peak if peak > 0 else audio

def trim_silence(audio, thresh=0.02):
    idx = np.where(np.abs(audio) > thresh)[0]
    if len(idx) == 0:
        return audio
    return audio[idx[0]:idx[-1]]

def fix_duration(audio, sr=TARGET_SR, duration=WINDOW_SEC):
    target_len = int(sr * duration)
    if len(audio) > target_len:
        return audio[:target_len]
    return np.pad(audio, (0, target_len - len(audio)))

def generate_log_mel(audio, sr=TARGET_SR):
    mel = librosa.feature.melspectrogram(
        y=audio,
        sr=sr,
        n_fft=2048,
        hop_length=512,
        win_length=2048,
        n_mels=N_MELS,
        power=2.0
    )
    mel_db = librosa.power_to_db(mel, ref=np.max)
    mel_db = (mel_db - mel_db.mean()) / (mel_db.std() + EPS)
    return mel_db

def fix_mel_frames(mel):
    if mel.shape[1] < TARGET_FRAMES:
        return np.pad(mel, ((0, 0), (0, TARGET_FRAMES -
mel.shape[1])))
    return mel[:, :TARGET_FRAMES]

def extract_features(y, sr=TARGET_SR):
    y = stereo_to_mono(y)
    y = peak_normalize(y)
    y = trim_silence(y)
    y = fix_duration(y, sr)

    mel = generate_log_mel(y, sr)
    mel = fix_mel_frames(mel)
    return mel

# ========================
# SLIDING WINDOWS
# ========================
```

```python
def sliding_windows(y, sr):
    win_len = int(sr * WINDOW_SEC)
    hop_len = int(sr * HOP_SEC)
    for start in range(0, len(y) - win_len + 1, hop_len):
        yield y[start:start + win_len]
def top_n_prediction(probs, n=1):
    pred = np.zeros_like(probs, dtype=int)
    top_idx = np.argsort(probs)[-n:]
    pred[top_idx] = 1
    return pred

# ========================
# TOP-K MEAN AGGREGATION
# ========================
def topk_mean_agg(audio_path, k=3):
    y, sr = librosa.load(audio_path, sr=TARGET_SR, mono=False)
    y = stereo_to_mono(y)

    window_probs = []

    for window in sliding_windows(y, sr):
        mel = extract_features(window, sr)
        mel = mel[..., np.newaxis][np.newaxis, ...]
        probs = model.predict(mel, verbose=0)[0]
        window_probs.append(probs)

    window_probs = np.array(window_probs)

    return np.mean(
        np.sort(window_probs, axis=0)[-k:], axis=0
    )

# ========================
# CLASS-WISE THRESHOLDING
# ========================
def apply_classwise_threshold(probs):
    pred = np.zeros(NUM_CLASSES, dtype=int)
    for i, cls in enumerate(CLASS_NAMES):
        pred[i] = int(probs[i] >= CLASS_THRESHOLDS[cls])
    return pred

def hybrid_prediction(probs, primary_k=1, secondary_thresh=0.6):
    pred = np.zeros_like(probs, dtype=int)

    # Primary instrument
    top_idx = np.argsort(probs)[-primary_k:]
    pred[top_idx] = 1

    # Secondary instruments (very strict)
    pred |= (probs >= secondary_thresh).astype(int)
```

```
    return pred

# ==========================
# LOAD MULTI-LABEL GT
# ==========================
def load_multilabel_gt(txt_path):
    labels = np.zeros(NUM_CLASSES)
    with open(txt_path, "r") as f:
        for inst in f.read().strip().split("\n"):
            if inst in class_to_id:
                labels[class_to_id[inst]] = 1
    return labels

# ==========================
# RANDOM 500-FILE EVALUATION
# ==========================
all_test_files = [f for f in os.listdir(TEST_AUDIO_ROOT) if
f.endswith(".wav")]

random.seed(RANDOM_SEED)
test_files = random.sample(all_test_files, NUM_TEST_FILES)

print(f"Evaluating on {len(test_files)} random test files")

y_test_true = []
y_test_pred = []

for wav in tqdm(test_files, desc="Inference"):
    audio_path = os.path.join(TEST_AUDIO_ROOT, wav)
    txt_path = audio_path.replace(".wav", ".txt")

    gt = load_multilabel_gt(txt_path)
    probs = topk_mean_agg(audio_path, k=3)
    pred = hybrid_prediction(probs)

    y_test_true.append(gt)
    y_test_pred.append(pred)

y_test_true = np.stack(y_test_true)
y_test_pred = np.stack(y_test_pred)

print("Evaluation shape:", y_test_true.shape)

# ==========================
# METRICS
# ==========================
subset_acc = accuracy_score(y_test_true, y_test_pred)
hamming_acc = 1.0 - hamming_loss(y_test_true, y_test_pred)
```

```
print(f"Subset Accuracy (Exact Match): {subset_acc:.4f}")
print(f"Hamming Accuracy: {hamming_acc:.4f}")
print("Micro F1 :", f1_score(y_test_true, y_test_pred,
average="micro"))
print("Macro F1 :", f1_score(y_test_true, y_test_pred,
average="macro"))

print(
    classification_report(
        y_test_true,
        y_test_pred,
        target_names=CLASS_NAMES,
        zero_division=0
    )
)
```

```
PROJECT_ROOT: C:\Games\Instrunet
```

```
Model: "functional"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ input_layer (InputLayer)             │ (None, 128, 126, 1)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d (Conv2D)                      │ (None, 128, 126, 32)        │
832 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ batch_normalization                  │ (None, 128, 126, 32)        │
128 │
│ (BatchNormalization)                 │                             │
│
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ activation (Activation)              │ (None, 128, 126, 32)        │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ max_pooling2d (MaxPooling2D)         │ (None, 64, 63, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ conv2d_1 (Conv2D)                    │ (None, 64, 63, 64)          │
51,264 │
```

```
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| batch_normalization_1               | (None, 64, 63, 64)          |
256 |
| (BatchNormalization)                |                             |
|
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| activation_1 (Activation)           | (None, 64, 63, 64)          |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| max_pooling2d_1 (MaxPooling2D)      | (None, 32, 31, 64)          |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| conv2d_2 (Conv2D)                   | (None, 32, 31, 128)         |
204,928 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| batch_normalization_2               | (None, 32, 31, 128)         |
512 |
| (BatchNormalization)                |                             |
|
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| activation_2 (Activation)           | (None, 32, 31, 128)         |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| max_pooling2d_2 (MaxPooling2D)      | (None, 16, 15, 128)         |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| conv2d_3 (Conv2D)                   | (None, 16, 15, 256)         |
819,456 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| batch_normalization_3               | (None, 16, 15, 256)         |
1,024 |
| (BatchNormalization)                |                             |
|
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| activation_3 (Activation)           | (None, 16, 15, 256)         |
0 |
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
| max_pooling2d_3 (MaxPooling2D)      | (None, 8, 7, 256)           |
```

```
0 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ global_average_pooling2d            │ (None, 256)                 │
0 │
│ (GlobalAveragePooling2D)            │                             │
│
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dense (Dense)                       │ (None, 256)                 │
65,792 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dropout (Dropout)                   │ (None, 256)                 │
0 │
├─────────────────────────────────────┼─────────────────────────────┼──────────
──────┤
│ dense_1 (Dense)                     │ (None, 11)                  │
2,827 │
└─────────────────────────────────────┴─────────────────────────────┴──────────
──────┘

 Total params: 2,293,080 (8.75 MB)

 Trainable params: 1,146,059 (4.37 MB)

 Non-trainable params: 960 (3.75 KB)

 Optimizer params: 1,146,061 (4.37 MB)

Evaluating on 500 random test files

Inference: 100%|██████████| 500/500 [14:17<00:00,  1.72s/it]

Evaluation shape: (500, 11)
Subset Accuracy (Exact Match): 0.5360
Hamming Accuracy: 0.9175
Micro F1 : 0.5504950495049505
Macro F1 : 0.3192140229533271
              precision    recall  f1-score   support

         cel       0.00      0.00      0.00         3
         cla       0.00      0.00      0.00         5
         flu       0.00      0.00      0.00         6
         gac       0.30      0.90      0.45        39
         gel       0.40      0.45      0.42        47
         org       0.00      0.00      0.00        26
         pia       0.80      0.38      0.52        84
         sax       0.60      0.73      0.66        59
         tru       0.50      0.16      0.24        31
         vio       0.48      0.37      0.42        27
```

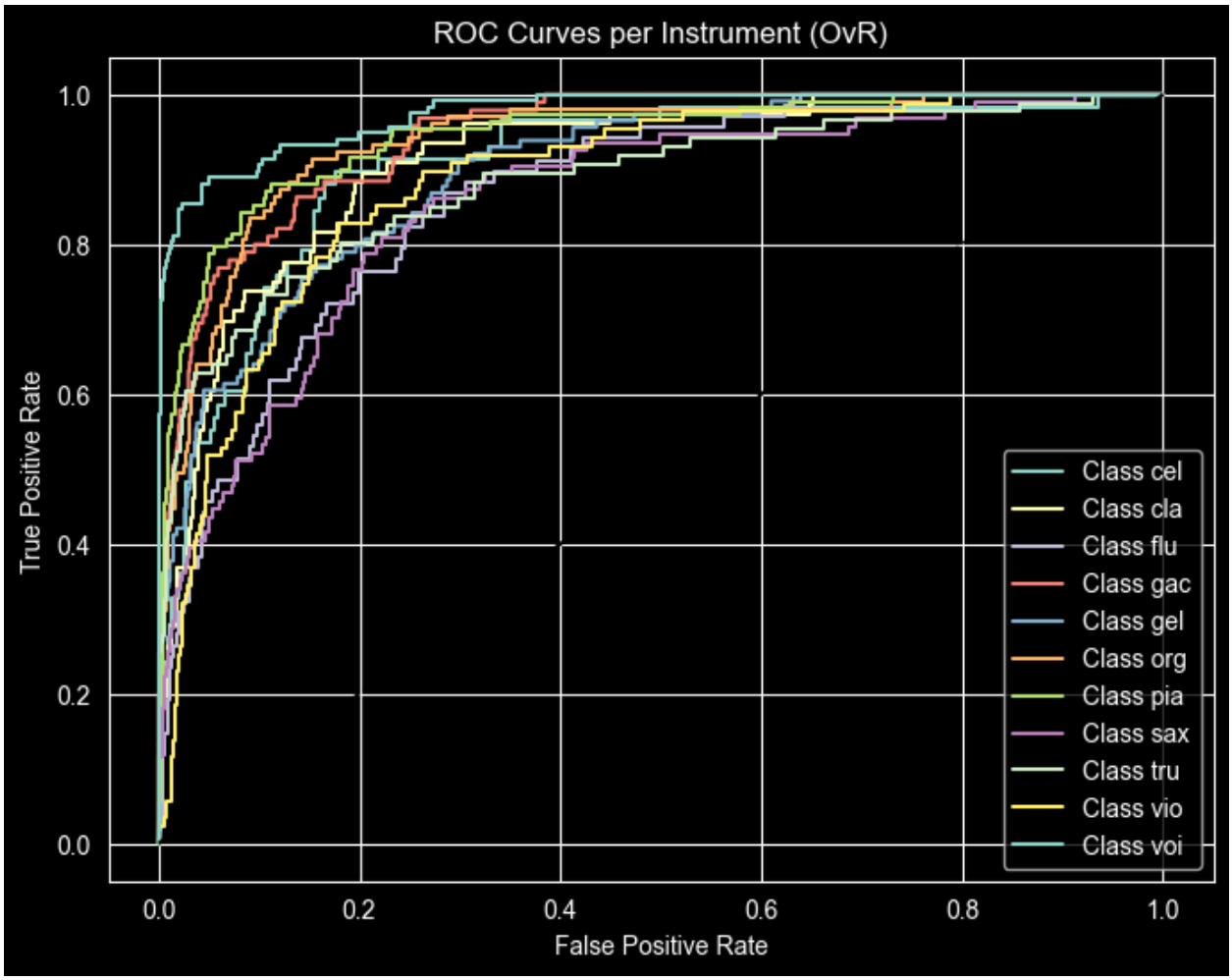
```
         voi       0.87      0.76      0.81       173

   micro avg       0.55      0.56      0.55       500
   macro avg       0.36      0.34      0.32       500
weighted avg       0.62      0.56      0.56       500
 samples avg       0.55      0.56      0.55       500
```

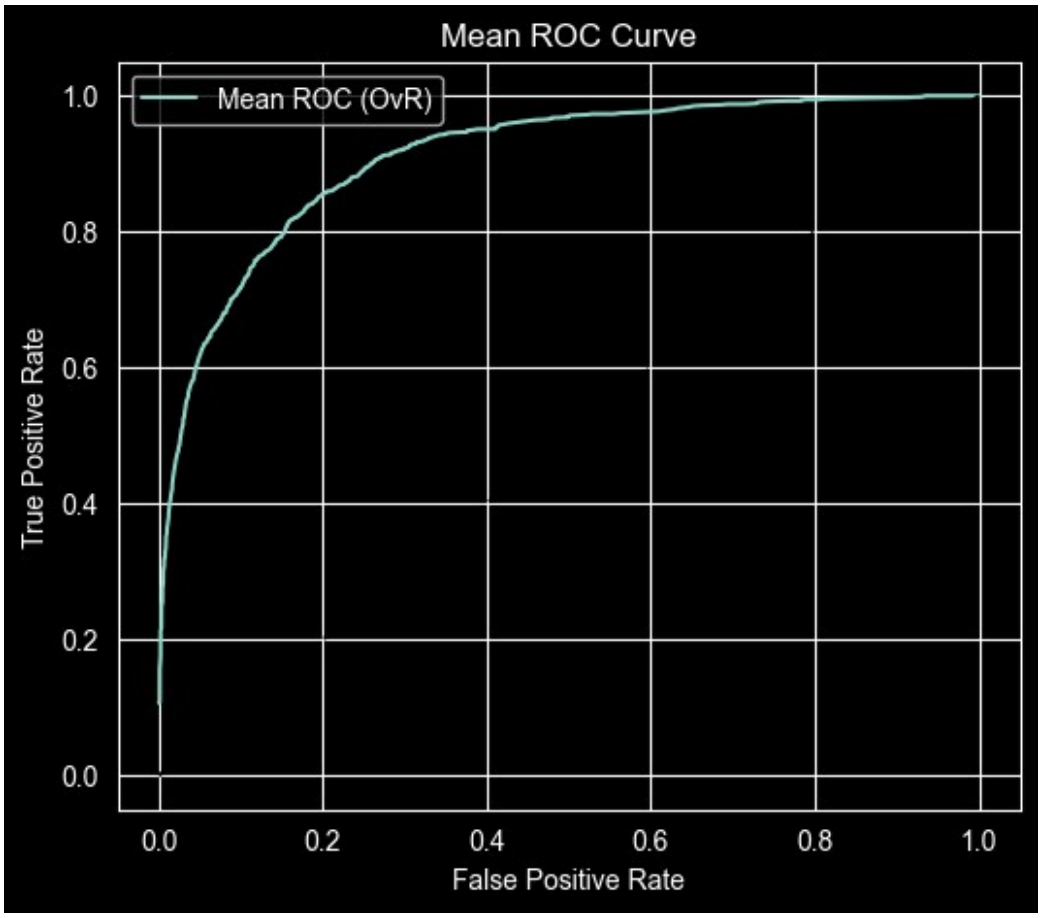
```python
##TASK 8: MERTICS EVALUATION
import os
import librosa
import librosa.display
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import pandas as pd
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    roc_auc_score,
    roc_curve
)
from sklearn.preprocessing import label_binarize
from pathlib import Path

# ======================================
# PROJECT ROOT (works in .py & Jupyter)
# ======================================
try:
    PROJECT_ROOT = Path(__file__).resolve().parent.parent
except NameError:
    # Jupyter notebook fallback
    PROJECT_ROOT = Path.cwd().parent

PROJECT_ROOT = PROJECT_ROOT.resolve()

print("PROJECT_ROOT:", PROJECT_ROOT)

# ======================================
# PATHS (STREAMLIT SAFE)
# ======================================
TEST_AUDIO_ROOT = PROJECT_ROOT / "Data" / "Testing" / "IRMAS-
TestingData-Part1" / "Part1"
MODEL_PATH = PROJECT_ROOT / "Model" /
"instrunet_irmas_sgd_finetuned.keras"
accuracy = accuracy_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
f1_macro = f1_score(y_true, y_pred, average="macro")
f1_micro = f1_score(y_true, y_pred, average="micro")
f1_weighted = f1_score(y_true, y_pred, average="weighted")

print(f"F1 Macro    : {f1_macro:.4f}")
print(f"F1 Micro    : {f1_micro:.4f}")
print(f"F1 Weighted : {f1_weighted:.4f}")
roc_auc = roc_auc_score(
    y_test,          # one-hot true labels
    y_prob,          # predicted probabilities
    average="macro",
    multi_class="ovr"
)

print(f"ROC-AUC (macro, OvR): {roc_auc:.4f}")
n_classes = y_test.shape[1]

# Binarized true labels
y_test_bin = label_binarize(y_true, classes=range(n_classes))

plt.figure(figsize=(8, 6))

for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_prob[:, i])
    plt.plot(fpr, tpr, label=f"Class {id_to_class[i]}")

plt.plot([0, 1], [0, 1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves per Instrument (OvR)")
plt.legend()
plt.grid(True)
plt.show()
fpr = dict()
tpr = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_prob[:, i])

all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)

for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

mean_tpr /= n_classes

plt.figure(figsize=(6, 5))
plt.plot(all_fpr, mean_tpr, label="Mean ROC (OvR)")
```

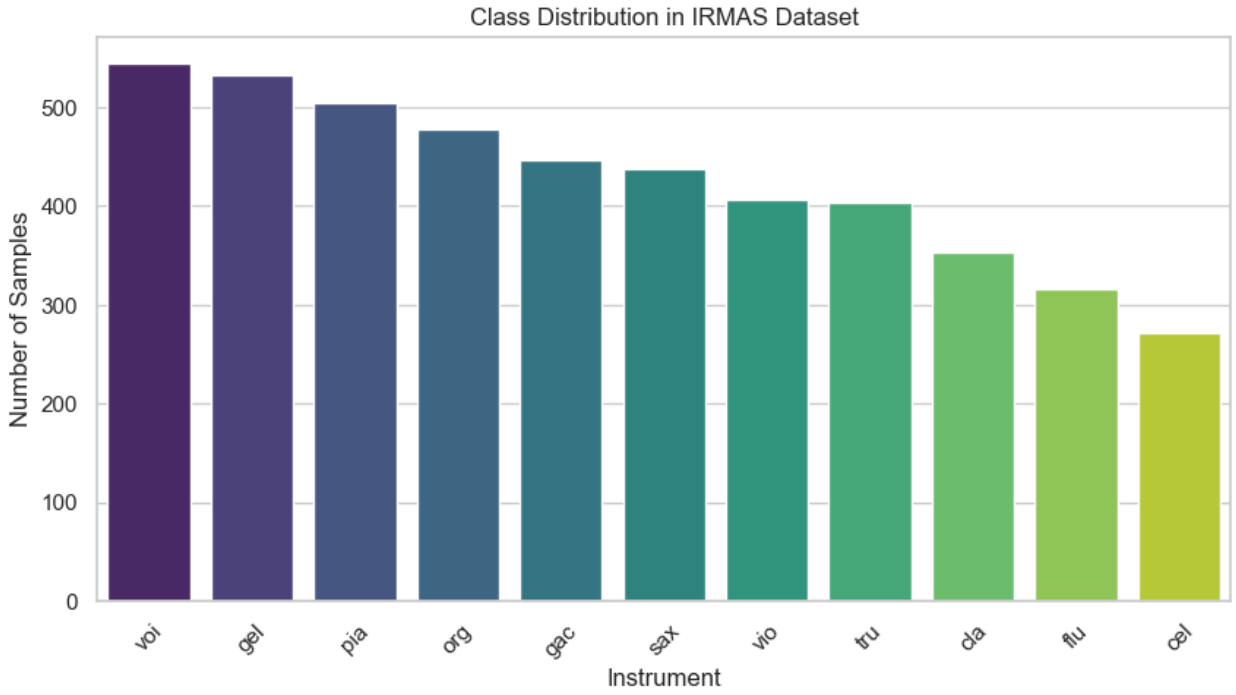
```
plt.plot([0, 1], [0, 1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Mean ROC Curve")
plt.legend()
plt.grid(True)
plt.show()

# Probabilities from model
y_prob = model.predict(X_test)

# Predicted class indices
y_pred = np.argmax(y_prob, axis=1)

# True class indices
y_true = np.argmax(y_test, axis=1)


sns.set(style="whitegrid")

DATASET_ROOT = PROJECT_ROOT / "Data" / "IRMAS_Mono_files" / "train"
SR = 16000
# %%
instrument_folders = [
    d for d in os.listdir(DATASET_ROOT)
    if os.path.isdir(os.path.join(DATASET_ROOT, d))
]

print("Number of instrument classes:", len(instrument_folders))
print("Instrument classes:", instrument_folders)
# %%
class_counts = {}

for inst in instrument_folders:
    inst_path = os.path.join(DATASET_ROOT, inst)
    class_counts[inst] = len([
        f for f in os.listdir(inst_path)
        if f.endswith(".wav")
    ])

df_counts = pd.DataFrame.from_dict(
    class_counts, orient="index", columns=["num_samples"]
).sort_values("num_samples", ascending=False)

df_counts
# %%
plt.figure(figsize=(10, 5))
sns.barplot(
    x=df_counts.index,
    y=df_counts["num_samples"],
```

```
    palette="viridis"
)
plt.title("Class Distribution in IRMAS Dataset")
plt.xlabel("Instrument")
plt.ylabel("Number of Samples")
plt.xticks(rotation=45)
plt.show()


PROJECT_ROOT: C:\Games\Instrunet
Accuracy: 0.5199
F1 Macro    : 0.4834
F1 Micro    : 0.5199
F1 Weighted : 0.5008
ROC-AUC (macro, OvR): 0.9094
```

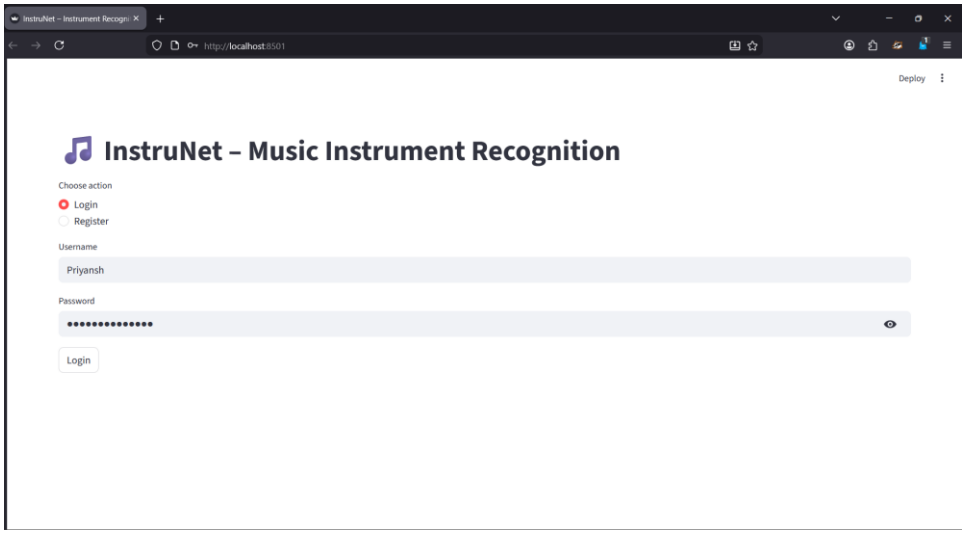


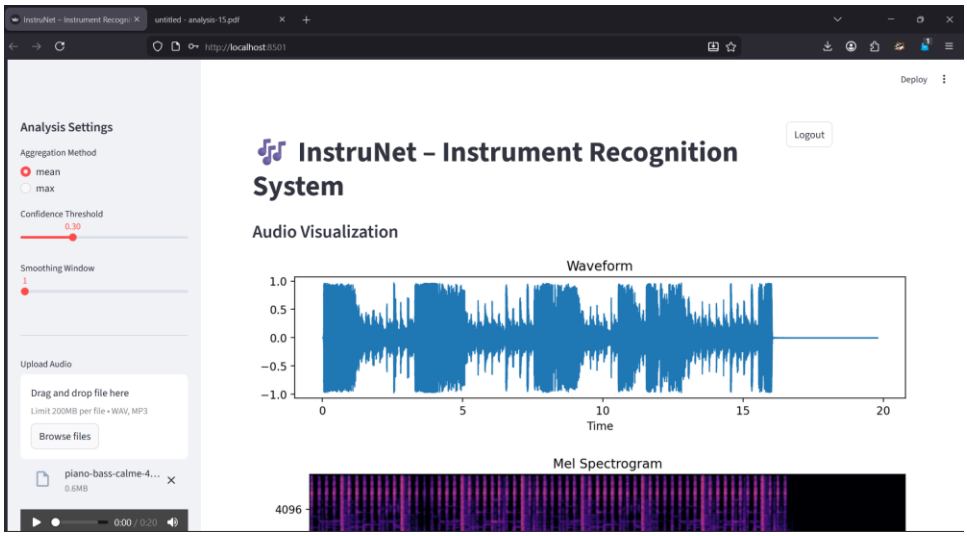


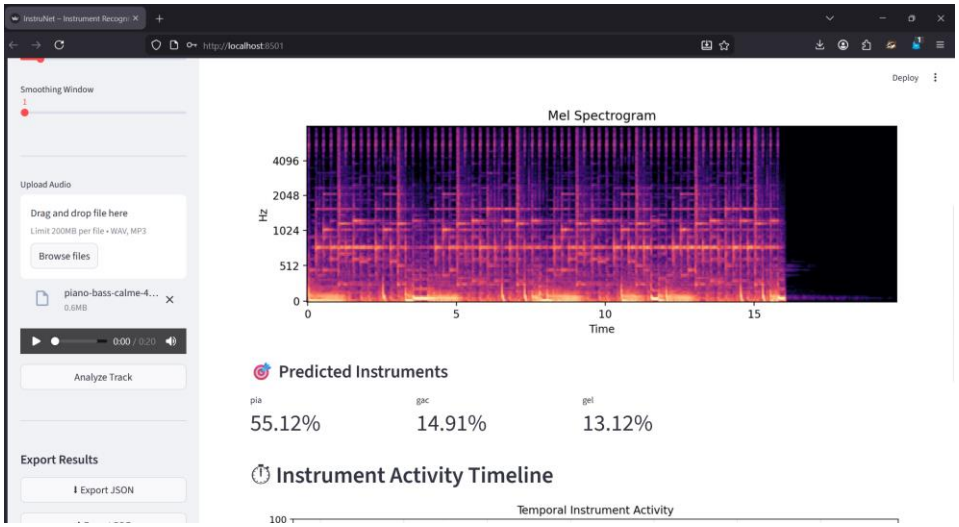

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 100ms/step
Number of instrument classes: 11
Instrument classes: ['cel', 'cla', 'flu', 'gac', 'gel', 'org', 'pia',
'sax', 'tru', 'vio', 'voi']

C:\Users\Priyansh\AppData\Local\Temp\
ipykernel_47176\3392558755.py:134: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(
```

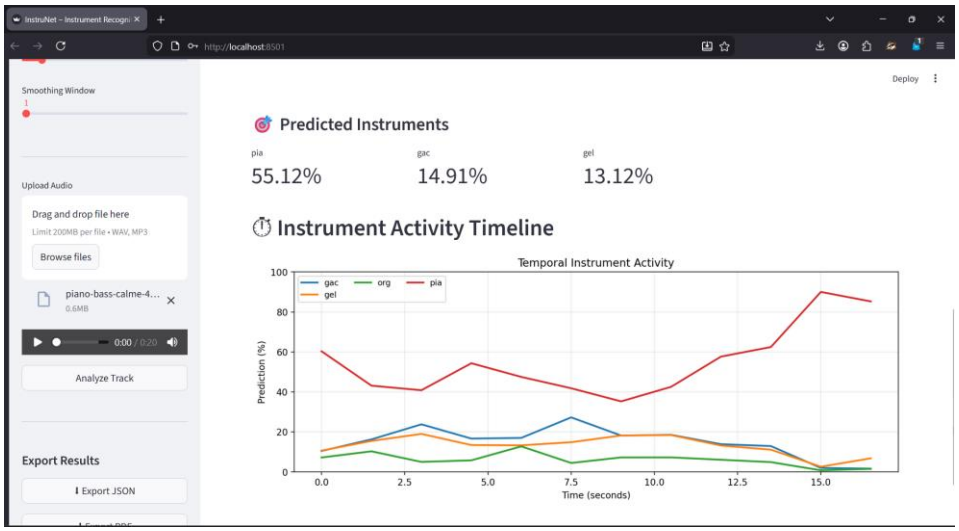

Class Distribution in IRMAS Dataset
Number of Samples
0
100
200
300
400
500
voi
gel
pia
org
gac
sax
vio
tru
cla
flu
cel
Instrument

WEBAPP SCREENSHOTS









JSON REPORT

"aggregation_method": "mean",

"smoothing_window": 1,

"threshold": 0.1,

"overall_detected_instruments": {

"gac": 14.912347793579102,

"gel": 13.118569374084473,

"pia": 55.12031936645508

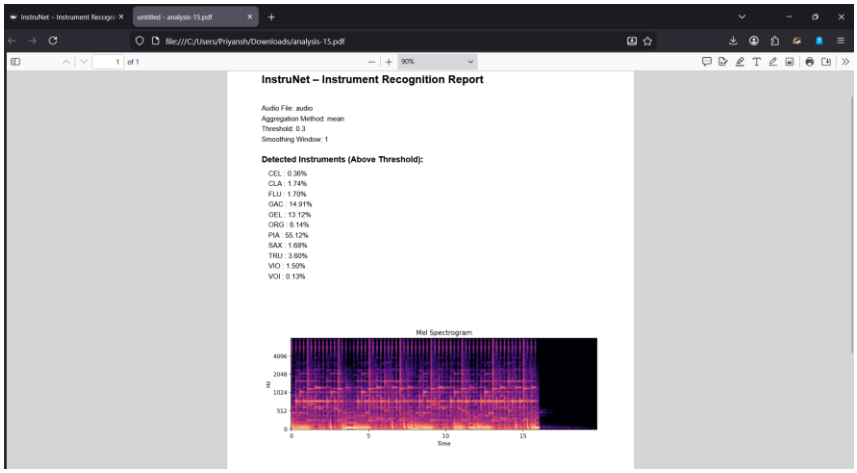
```
},
"per_hop_activity": [
 {
  "time": 0.0,
  "active_instruments": {
   "gac": 10.496000289916992,
   "gel": 10.68593692779541,
   "pia": 60.352237701416016
  }
 },
 {
  "time": 1.5,
  "active_instruments": {
   "gac": 16.286550521850586,
   "gel": 15.517394065856934,
   "org": 10.326424598693848,
   "pia": 43.15079879760742
  }
 },
 {
  "time": 3.0,
  "active_instruments": {
   "gac": 23.817052841186523,
   "gel": 19.036176681518555,
   "pia": 40.86888885498047
```

```
  }
},
{
  "time": 4.5,
  "active_instruments": {
    "gac": 16.697595596313477,
    "gel": 13.463744163513184,
    "pia": 54.371253967285156
  }
},
{
  "time": 6.0,
  "active_instruments": {
    "gac": 17.027881622314453,
    "gel": 13.315563201904297,
    "org": 12.790470123291016,
    "pia": 47.5260124206543
  }
},
{
  "time": 7.5,
  "active_instruments": {
    "gac": 27.300247192382812,
    "gel": 14.881040573120117,
    "pia": 41.85485076904297
```

```
  }
},
{
  "time": 9.0,
  "active_instruments": {
    "gac": 18.21084976196289,
    "gel": 18.24799919128418,
    "pia": 35.29325866699219
  }
},
{
  "time": 10.5,
  "active_instruments": {
    "gac": 18.579923629760742,
    "gel": 18.4548282623291,
    "pia": 42.56599426269531
  }
},
{
  "time": 12.0,
  "active_instruments": {
    "gac": 13.896539688110352,
    "gel": 13.209654808044434,
    "pia": 57.68319320678711
  }
```

```
    },
    {
      "time": 13.5,
      "active_instruments": {
        "gac": 12.951148986816406,
        "gel": 11.152650833129883,
        "pia": 62.47334289550781
      }
    },
    {
      "time": 15.0,
      "active_instruments": {
        "pia": 90.0209732055664
      }
    },
    {
      "time": 16.5,
      "active_instruments": {
        "pia": 85.28304290771484
      }
    }
  ]
}
```


PDF REPORT


InstruNet – Instrument Recogn
untitled - analysis-15.pdf
file:///C:/Users/Priyansh/Downloads/analysis-15.pdf
1 of 1
90%
InstruNet – Instrument Recognition Report
Audio File: audio
Aggregation Method: mean
Threshold: 0.3
Smoothing Window: 1
Detected Instruments (Above Threshold):
CEL : 0.36%
CLA : 1.74%
FLU : 1.70%
GAC : 14.91%
GEL : 13.12%
ORG : 6.14%
PIA : 55.12%
SAX : 1.68%
TRU : 3.60%
VIO : 1.50%
VOI : 0.13%
Mel Spectrogram
4096
2048
1024
512
0
Hz
0
5
10
15
Time