

```

from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

# Paths to your small subset folders
dev_path = "/content/drive/MyDrive/Papers/Paper_6/Datasets/dev"
train_path = "/content/drive/MyDrive/Papers/Paper_6/Datasets/train"
test_path = "/content/drive/MyDrive/Papers/Paper_6/Datasets/test"

# Example: list first 5 files from each folder
import os

print("Dev samples:", os.listdir(dev_path)[:5])
print("Train samples:", os.listdir(train_path)[:5])
print("Test samples:", os.listdir(test_path)[:5])

Dev samples: ['004053.mp4', '004054.mp4', '004055.mp4', '004056.mp4', '004057.mp4']
Train samples: ['000469.mp4', '000471.mp4', '000470.mp4', '000472.mp4', '000473.mp4']
Test samples: ['000000.mp4', '000001.mp4', '000002.mp4', '000004.mp4', '000003.mp4']

```

```

pip install mtcnn opencv-python-headless tqdm pandas soundfile

Collecting mtcnn
  Downloading mtcnn-1.0.0-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.12/dist-packages (4.12.0.88)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (4.67.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: soundfile in /usr/local/lib/python3.12/dist-packages (0.13.1)
Requirement already satisfied: joblib>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from mtcnn) (1.5.2)
Collecting lz4<4.3.3 (from mtcnn)
  Downloading lz4-4.4.4-cp312-cp312-manylinux_2_17_x86_64.whl.metadata (3.8 kB)
Requirement already satisfied: numpy<2.3.0,>=2 in /usr/local/lib/python3.12/dist-packages (from opencv-python-headless) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.12/dist-packages (from soundfile) (2.0.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.0->soundfile) (2.23)
Requirement already satisfied: sixx=>1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Downloading mtcnn-1.0.0-py3-none-any.whl (1.9 MB)
  1.9/1.9 MB 57.1 MB/s eta 0:00:00
Downloaded lz4-4.4.4-cp312-cp312-manylinux_2_17_x86_64.whl (1.3 MB)
  1.3/1.3 MB 58.8 MB/s eta 0:00:00
Installing collected packages: lz4, mtcnn
Successfully installed lz4-4.4.4 mtcnn-1.0.0

```

```

import tensorflow as tf
from mtcnn import MTCNN
import pandas as pd
import cv2

print("Tensorflow version:", tf.__version__)
print("tf.keras available:", hasattr(tf, 'keras'))
print("MTCNN imported successfully!")

TensorFlow version: 2.19.0
tf.keras available: True
MTCNN imported successfully!

import os
import cv2
from mtcnn import MTCNN
from tqdm import tqdm

# --- Paths ---
dataset_root = "/content/drive/My Drive/Papers/Paper_6/Datasets/"
output_root = "/content/drive/My Drive/Papers/Paper_6/Datasets/Faces_Extracted/"

subfolders = ['dev', 'train', 'test']

detector = MTCNN()

```

```

for sub in subfolders:
    input_folder = os.path.join(dataset_root, sub)
    output_folder = os.path.join(output_root, sub)
    os.makedirs(output_folder, exist_ok=True)

videos = [f for f in os.listdir(input_folder) if f.endswith('.mp4')]

for vid_file in tqdm(videos, desc=f"Processing {sub}"):
    vid_path = os.path.join(input_folder, vid_file)
    cap = cv2.VideoCapture(vid_path)
    frame_count = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Detect faces
        faces = detector.detect_faces(frame)
        for i, face in enumerate(faces):
            x, y, w, h = face['box']
            x, y = max(0, x), max(0, y)
            face_crop = frame[y:y+h, x:x+w]
            face_filename = f"{os.path.splitext(vid_file)[0]}_frame{frame_count}_face{i}.jpg"
            cv2.imwrite(os.path.join(output_folder, face_filename), face_crop)

        frame_count += 1

cap.release()

Processing dev: 100%|██████████| 5/5 [03:06<00:00, 37.34s/it]
Processing train: 100%|██████████| 7/7 [03:10<00:00, 27.27s/it]
Processing test: 100%|██████████| 5/5 [01:54<00:00, 22.81s/it]

```

---

```

import os
import librosa
import numpy as np
import json
from tqdm import tqdm

# --- Paths ---
video_root = "/content/drive/MyDrive/Papers/Paper 6/Datasets" # <- updated path
output_root = os.path.join(video_root, "MFCC_Features")
os.makedirs(output_root, exist_ok=True)

# --- MFCC Configuration ---
sr = 16000 # sampling rate
n_mfcc = 13 # number of MFCCs
hop_length = 512 # hop length
n_fft = 2048 # FFT window

# --- Initialize metadata ---
metadata_info = {}

# --- Process each split ---
splits = ["dev", "train", "test"]

for split in splits:
    split_path = os.path.join(video_root, split)
    videos = [f for f in os.listdir(split_path) if f.endswith(".mp4")]
    metadata_info[split] = []

    for video_file in tqdm(videos, desc=f"Processing {split}"):
        video_path = os.path.join(split_path, video_file)
        y, sr = librosa.load(video_path, sr=sr) # load audio from video
        mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc, hop_length=hop_length, n_fft=n_fft)

        # Save MFCC as numpy file
        mfcc_filename = f"{os.path.splitext(video_file)[0]}_mfcc.npy"
        mfcc_path = os.path.join(output_root, mfcc_filename)
        np.save(mfcc_path, mfcc)

        # Update metadata
        metadata_info[split].append({
            "video": video_file,
            "mfcc_path": mfcc_path,
            "mfcc_shape": mfcc.shape
        })

# --- Save metadata as JSON ---
metadata_json_path = os.path.join(output_root, "mfcc_metadata.json")
with open(metadata_json_path, "w") as f:

```

```

    with open(metadata_json_path, "w") as f:
        json.dump(metadata_info, f, indent=4)
        print("MFCC extraction completed! Metadata saved.")

    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing train: 43% [██████| 3/7 [00:00<00:01, 3.52it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing train: 57% [█████| 4/7 [00:01<00:01, 3.54it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing train: 71% [█████| 5/7 [00:01<00:01, 3.62it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing train: 86% [████████| 6/7 [00:01<00:01, 4.07it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing train: 100% [████████| 7/7 [00:01<00:01, 3.99it/s]
    Processing test: 0% [ 0/5 [00:00<00:00, 0.00it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing test: 20% [██| 1/5 [00:00<00:00, 4.91it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing test: 40% [███| 2/5 [00:00<00:00, 5.46it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing test: 60% [████| 3/5 [00:00<00:00, 5.63it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing test: 80% [█████| 4/5 [00:00<00:00, 5.72it/s]/tmp/ipython-input-1462058705.py:31: UserWarning: PySoundFile failed. Trying audioread instead.
    y, _ = librosa.load(video_path, sr=sr) # load audio from video
    /usr/local/lib/python3.12/dist-packages/librosa/core/audio.py:184: FutureWarning: librosa.core.audio._audioread_load
        DeprecationWarning: as of librosa version 0.10.0.
        It will be removed in librosa version 1.0.
    y, sr_native = _audioread_load(path, offset, duration, dtype)
    Processing test: 100% [████████| 5/5 [00:00<00:00, 5.60it/s]MFCC extraction completed! Metadata saved.


```

```

import numpy as np
import os

output_root = "/content/drive/MyDrive/Papers/Paper_6/Datasets/MFCC_Features"
files = os.listdir(output_root)
for f in files[:5]: # show first 5 files
    if f.endswith(".npy"):
        mfcc = np.load(os.path.join(output_root, f))
        print(f"\t{f}: shape = {mfcc.shape}")


```

```

import os
import numpy as np

# --- Paths ---
faces_root = "/content/drive/MyDrive/Papers/Paper 6/Datasets/Faces_Extracted"
mfcc_root = "/content/drive/MyDrive/Papers/Paper 6/Datasets/MFCC_Features"

# --- Load MFCC features ---
mfcc_files = [f for f in os.listdir(mfcc_root) if f.endswith(".npy")]
mfcc_data = {}

for f in mfcc_files:
    video_id = f.split("_mfcc.npy")[0]
    mfcc_path = os.path.join(mfcc_root, f)
    mfcc_data[video_id] = np.load(mfcc_path)
    print(f"{f}: shape = {mfcc_data[video_id].shape}")

# --- Verify matching with face folders ---
for split in ["dev", "train", "test"]:
    split_path = os.path.join(faces_root, split)
    if os.path.exists(split_path):
        videos = sorted(os.listdir(split_path))
        print(f"{split} samples detected: {len(videos)}")
    else:
        print(f"{split} folder not found in {faces_root}")

```

---

```

004053_mfcc.npy: shape = (13, 239)
004054_mfcc.npy: shape = (13, 249)
004055_mfcc.npy: shape = (13, 243)
004056_mfcc.npy: shape = (13, 247)
004057_mfcc.npy: shape = (13, 181)
0040469_mfcc.npy: shape = (13, 159)
000471_mfcc.npy: shape = (13, 163)
000470_mfcc.npy: shape = (13, 185)
000472_mfcc.npy: shape = (13, 189)
000473_mfcc.npy: shape = (13, 129)
000474_mfcc.npy: shape = (13, 157)
000475_mfcc.npy: shape = (13, 133)
000000_mfcc.npy: shape = (13, 145)
000001_mfcc.npy: shape = (13, 131)
000002_mfcc.npy: shape = (13, 149)
000004_mfcc.npy: shape = (13, 147)
000003_mfcc.npy: shape = (13, 135)
dev samples detected: 918
train samples detected: 822
test samples detected: 551

```

---

```

import os
import numpy as np
import cv2
from tqdm import tqdm

# --- Paths ---
faces_root = "/content/drive/MyDrive/Papers/Paper 6/Datasets/Faces_Extracted"
mfcc_root = "/content/drive/MyDrive/Papers/Paper 6/Datasets/MFCC_Features"

# --- Parameters ---
num_mfcc = 13 # matches your MFCC extraction

# --- Simple baseline evaluation ---
results = {}

for split in ["dev", "train", "test"]:
    split_faces = os.path.join(faces_root, split)
    videos = sorted(os.listdir(split_faces))
    print(f"\nProcessing {split}: {len(videos)} videos")

    for vid in tqdm(videos):
        video_id = os.path.splitext(vid)[0]

        # Load MFCC
        mfcc_path = os.path.join(mfcc_root, f"{video_id}_mfcc.npy")
        if not os.path.exists(mfcc_path):
            continue
        mfcc = np.load(mfcc_path) # shape (13, T)

        # Load first face frame as dummy visual feature
        face_frame_path = os.path.join(split_faces, vid)
        cap = cv2.VideoCapture(face_frame_path)
        ret, frame = cap.read()
        cap.release()
        if not ret:
            continue

```

```

if not ret:
    continue
frame_feature = cv2.resize(frame, (64, 64)) # downsample for baseline
frame_feature = frame_feature.mean(axis=2) # grayscale
frame_feature = frame_feature.flatten() # simple 1D vector

# Dummy similarity: cosine similarity between MFCC mean and frame mean
mfcc_mean = mfcc.mean(axis=1)
frame_mean = frame_feature.mean()
similarity = np.dot(mfcc_mean, frame_mean) / (np.linalg.norm(mfcc_mean) * (frame_mean + 1e-6))

results[video_id] = similarity

# --- Show some baseline results ---
for i, (vid, score) in enumerate(results.items()):
    print(f"(vid): similarity = {score:.4f}")
    if i > 9:
        break

```

Processing dev: 918 videos  
100% [██████████] 918/918 [00:00<00:00, 4274.77it/s]

Processing train: 822 videos  
100% [██████████] 822/822 [00:00<00:00, 3763.18it/s]

Processing test: 551 videos  
100% [██████████] 551/551 [00:00<00:00, 3754.45it/s]

---

```

import os
import numpy as np
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# --- Paths ---
mfcc_root = "/content/drive/MyDrive/Papers/Paper 6/Datasets/MFCC_Features"

# --- Collect video names ---
video_files = [f for f in os.listdir(mfcc_root) if f.endswith("_mfcc.npy")]
video_names = [f.replace("_mfcc.npy", "") for f in video_files]
print(f"Detected {len(video_names)} MFCC files.")

# --- Assign labels (first half real, second half fake) ---
label_mapping = {}
half = len(video_names) // 2
for i, name in enumerate(video_names):
    label_mapping[name] = 0 if i < half else 1

# --- Determine maximum number of frames ---
max_frames = 0
for name in video_names:
    mfcc = np.load(os.path.join(mfcc_root, f"{name}_mfcc.npy"))
    if mfcc.shape[1] > max_frames:
        max_frames = mfcc.shape[1]

```

```

print(f"Maximum frames in any MFCC: {max_frames}")

# --- Prepare MFCC features with padding ---
n_mfcc = 13 # number of MFCC coefficients
X, y = [], []

for name in tqdm(video_names, desc="Loading MFCC features"):
    mfcc_file = os.path.join(mfcc_root, f"{name}_mfcc.npy")
    mfcc = np.load(mfcc_file) # shape (n_mfcc, n_frames)
    # Pad with zeros if shorter than max_frames
    if mfcc.shape[1] < max_frames:
        pad_width = max_frames - mfcc.shape[1]
        mfcc = np.pad(mfcc, ((0,0),(0,pad_width)), mode='constant')
    # Flatten for baseline
    X.append(mfcc.flatten())
    y.append(label_mapping[name])

X = np.array(X)
y = np.array(y)

print("Feature matrix shape:", X.shape)
print("Labels shape:", y.shape)

# --- Split into train/test ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

```

  

```

# --- Simple baseline model ---
clf = LogisticRegression(max_iter=500)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print(f"\nBaseline accuracy: {acc:.4f}")
print("\nClassification report:\n", classification_report(y_test, y_pred))

```

Detected 17 MFCC files.  
Maximum frames in any MFCC: 249  
Loading MFCC features: 100%|██████████| 17/17 [00:00<00:00, 266.05it/s]Feature matrix shape: (17, 3237)  
Labels shape: (17,)  
Train set: (13, 3237) (13,)  
Test set: (4, 3237) (4,)  
Baseline accuracy: 1.0000  
Classification report:  
precision recall f1-score support  
0 1.00 1.00 1.00 2  
1 1.00 1.00 1.00 2  
accuracy 1.00 1.00 1.00 4  
macro avg 1.00 1.00 1.00 4  
weighted avg 1.00 1.00 1.00 4