In the name of God.

Final Project

Epileptic Seizure Prediction Using EEG Signals or Video ¶

In [1]:
```
# Mount Google Drive to access EEG data
from google.colab import drive
drive.mount('/content/drive')

# Check the contents of the chb directory
import os

# Path
chb_path = '/content/drive/MyDrive/G_Pr/chb'

# List the folders (patients)
print("Available patient folders:")
print(os.listdir(chb_path))
```

```
Mounted at /content/drive
Available patient folders:
['chb02', 'chb04', 'chb10', 'chb01', 'chb08', 'chb09', 'chb06', 'chb05', 'chb07', 'chb03', 'chb20', 'chb16',
'chb19', 'chb18', 'chb15', 'chb11', 'chb14', 'chb12', 'chb13', 'chb17', 'chb21', 'chb24', 'chb23', 'chb22']
```

In [2]:
```
import glob

# List .edf files for each patient
for patient in os.listdir(chb_path):
    patient_folder = os.path.join(chb_path, patient)
    edf_files = sorted(glob.glob(os.path.join(patient_folder, "*.edf")))

    print(f"{patient}: {len(edf_files)} EDF files")
    for edf_file in edf_files[:3]:  # Just show first 3 as preview in each folder
        print("   ", os.path.basename(edf_file))
    print("   ...")
```

```
chb02: 36 EDF files
    chb02_01.edf
    chb02_02.edf
    chb02_03.edf
   ...
chb04: 42 EDF files
    chb04_01.edf
    chb04_02.edf
    chb04_03.edf
   ...
chb10: 25 EDF files
    chb10_01.edf
    chb10_02.edf
    chb10_03.edf
   ...
chb01: 42 EDF files
    chb01_01.edf
    chb01_02.edf
    chb01_03.edf
```

In [3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", fo
rce_remount=True).
```

In [4]:
```
# Install necessary packages
!pip install mne matplotlib numpy

# --- Load one EDF file to inspect EEG signals ---

import mne
import matplotlib.pyplot as plt

# Path to one sample EDF file (you can change the patient and file later)
edf_sample_path = os.path.join(chb_path, "chb01", "chb01_01.edf")

# Load the EDF file
raw = mne.io.read_raw_edf(edf_sample_path, preload=True)
print(raw)

# Plot EEG signals (first 30 seconds)
raw.plot(duration=30, n_channels=23, scalings='auto', show=True)
```

```
Collecting mne
  Downloading mne-1.9.0-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: decorator in /usr/local/lib/python3.11/dist-packages (from mne) (4.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from mne) (3.1.6)
Requirement already satisfied: lazy-loader>=0.3 in /usr/local/lib/python3.11/dist-packages (from mne) (0.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from mne) (24.2)
Requirement already satisfied: pooch>=1.5 in /usr/local/lib/python3.11/dist-packages (from mne) (1.8.2)
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.11/dist-packages (from mne) (1.15.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from mne) (4.67.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib)
(1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.
12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotli
b) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotli
b) (1.4.8)
```

In [5]:
```
# Load a specific EDF file and plot EEG signals

import mne
import matplotlib.pyplot as plt

# Path to a specific EDF file (e.g., chb01_01.edf from subject chb01)
edf_file = '/content/drive/MyDrive/G_Pr/chb/chb01/chb01_01.edf'

# Load the EEG data from the EDF file with preload=True to load it into memory
raw = mne.io.read_raw_edf(edf_file, preload=True)

# Display basic information about the EEG recording
print(raw.info)

# Plot the first 10 seconds of EEG data using 20 channels and automatic scaling
raw.plot(duration=10, n_channels=20, scalings='auto')
```

```
Extracting EDF parameters from /content/drive/MyDrive/G_Pr/chb/chb01/chb01_01.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 921599  =      0.000 ...  3599.996 secs...

<ipython-input-5-1e4e2f1c776f>:10: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8-
P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_file, preload=True)

<Info | 8 non-empty values
 bads: []
 ch_names: FP1-F7, F7-T7, T7-P7, P7-O1, FP1-F3, F3-C3, C3-P3, P3-O1, ...
 chs: 23 EEG
 custom_ref_applied: False
 highpass: 0.0 Hz
 lowpass: 128.0 Hz
 meas_date: 2076-11-06 11:42:54 UTC
 nchan: 23
 projs: []
```

In [6]:
```
# Extract raw EEG data and timestamps

# Get data and timestamps
eeg_data, times = raw.get_data(return_times=True)

print("EEG data shape:", eeg_data.shape)   # (channels, samples)
print("Timestamps shape:", times.shape)    # (samples,)
print("Sample rate:", raw.info['sfreq'])   # sampling frequency
```

```
EEG data shape: (23, 921600)
Timestamps shape: (921600,)
Sample rate: 256.0
```

In [7]:
```
import os
import mne

# Root directory containing the EEG data
root_dir = "/content/drive/MyDrive/G_Pr/chb"

# Band-pass filter range (1 to 40 Hz)
low_freq = 1.0
high_freq = 40.0

# Get the list of patient directories
patients = sorted([p for p in os.listdir(root_dir) if os.path.isdir(os.path.join(root_dir, p))])

# Iterate through all patients and their EDF files
for patient in patients:
    patient_path = os.path.join(root_dir, patient)
    edf_files = sorted([f for f in os.listdir(patient_path) if f.endswith('.edf')])

    print(f"\n Patient {patient} - {len(edf_files)} EDF files")

    for edf_file in edf_files:
        edf_path = os.path.join(patient_path, edf_file)
        print(f" Loading {edf_file}...")

        try:
            # Load the raw EDF file
            raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)

            # Apply band-pass filter from 1 to 40 Hz
            raw.filter(l_freq=low_freq, h_freq=high_freq, verbose=False)

            # Extract the data and corresponding timestamps
            data, timestamps = raw.get_data(return_times=True)
            print(f" Done. Data shape: {data.shape}")

        except Exception as e:
            print(f" Failed to process {edf_file}: {e}")
```

```
Patient chb01 - 42 EDF files
Loading chb01_01.edf...
```

```
<ipython-input-7-792999b4b363>:27: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8-
P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)

 Done. Data shape: (23, 921600)
 Loading chb01_02.edf...

<ipython-input-7-792999b4b363>:27: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8-
P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)

 Done. Data shape: (23, 921600)
 Loading chb01_03.edf...

<ipython-input-7-792999b4b363>:27: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8-
```

In [8]:
```
import os
import re

def parse_summary_file(summary_path):
    seizure_dict = {}
    current_edf = None

    with open(summary_path, 'r') as f:
        for line in f:
            line = line.strip()

            # Detect EDF file name
            if line.endswith('.edf'):
                current_edf = line
                seizure_dict[current_edf] = []

            # Detect seizure start/end
            elif 'Seizure' in line and 'Start Time' in line:
                start_time = float(re.findall(r'(\d+\.?\d*)', line)[0])
                seizure_dict[current_edf].append({'start': start_time})
            elif 'Seizure' in line and 'End Time' in line:
                end_time = float(re.findall(r'(\d+\.?\d*)', line)[0])
                if seizure_dict[current_edf]:
                    seizure_dict[current_edf][-1]['end'] = end_time

    return seizure_dict
```

In [9]:
```
# --- Read seizure info for all patients from their summary files ---

base_dir = "/content/drive/MyDrive/G_Pr/chb"
patient_folders = sorted([f for f in os.listdir(base_dir) if os.path.isdir(os.path.join(base_dir, f))])

all_seizure_data = {}

for patient_id in patient_folders:
    summary_path = os.path.join(base_dir, patient_id, f"{patient_id}-summary.txt")
    if os.path.exists(summary_path):
        print(f"Parsing: {summary_path}")
        seizure_info = parse_summary_file(summary_path)
        all_seizure_data[patient_id] = seizure_info
    else:
        print(f"Summary file not found for {patient_id}")

print("\n Finished parsing all patients.")
```

```
Parsing: /content/drive/MyDrive/G_Pr/chb/chb01/chb01-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb02/chb02-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb03/chb03-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb04/chb04-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb05/chb05-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb06/chb06-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb07/chb07-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb08/chb08-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb09/chb09-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb10/chb10-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb11/chb11-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb12/chb12-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb13/chb13-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb14/chb14-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb15/chb15-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb16/chb16-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb17/chb17-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb18/chb18-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb19/chb19-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb20/chb20-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb21/chb21-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb22/chb22-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb23/chb23-summary.txt
Parsing: /content/drive/MyDrive/G_Pr/chb/chb24/chb24-summary.txt

 Finished parsing all patients.
```

In [10]:
```
import os
import numpy as np
import pandas as pd
import mne

# Root directory
root_dir = '/content/drive/MyDrive/G_Pr/chb'

# Output list
all_features = []
expected_channels = None  # Number of channels to be used

for patient_folder in os.listdir(root_dir):
    patient_path = os.path.join(root_dir, patient_folder)
    if not os.path.isdir(patient_path):
        continue

    seizures = all_seizure_data.get(patient_folder, [])

    for edf_file in os.listdir(patient_path):
        if not edf_file.endswith('.edf'):
            continue

        edf_path = os.path.join(patient_path, edf_file)

        try:
            raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)
            data, times = raw.get_data(return_times=True)
        except Exception as e:
            print(f"Error reading {edf_file}: {e}")
            continue

        # Set expected channel count from first successful file
        if expected_channels is None:
            expected_channels = data.shape[0]

        # Skip files with different number of channels
        if data.shape[0] != expected_channels:
            print(f"Skipping {edf_file} due to mismatched channel count.")
            continue

        # Seizure detection
        contains_seizure = 0
```

```
        for item in seizures:
            if not isinstance(item, (list, tuple)) or len(item) != 3:
                print(f" Warning: unexpected seizure entry format: {item}")
                continue
            seizure_file, onset, duration = item
            if seizure_file == edf_file:
                contains_seizure = 1
                break

        # Extract features
        features = []
        for channel in data:
            features.extend([
                np.mean(channel),
                np.std(channel),
                np.sum(channel ** 2) / len(channel),
            ])

        features.append(contains_seizure)
        features.append(patient_folder)
        features.append(edf_file)

        all_features.append(features)

# Create column names
column_names = []
for ch in range(expected_channels):
    column_names.extend([f'ch{ch+1}_mean', f'ch{ch+1}_std', f'ch{ch+1}_energy'])
column_names += ['seizure', 'patient', 'filename']

# Convert to DataFrame
features_df = pd.DataFrame(all_features, columns=column_names)

# Preview
features_df.head()
```

```
<ipython-input-10-ed86186f89d1>:27: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8
-P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)
```

```
Warning: unexpected seizure entry format: File Name: chb02_01.edf
Warning: unexpected seizure entry format: File Name: chb02_02.edf
Warning: unexpected seizure entry format: File Name: chb02_03.edf
Warning: unexpected seizure entry format: File Name: chb02_04.edf
Warning: unexpected seizure entry format: File Name: chb02_05.edf
Warning: unexpected seizure entry format: File Name: chb02_06.edf
Warning: unexpected seizure entry format: File Name: chb02_07.edf
Warning: unexpected seizure entry format: File Name: chb02_08.edf
Warning: unexpected seizure entry format: File Name: chb02_09.edf
Warning: unexpected seizure entry format: File Name: chb02_10.edf
```

Initial setting

In [11]:
```
import os
import numpy as np
import mne
import pandas as pd
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pickle

# Parameters
WINDOW_SIZE = 10  # in seconds
SAMPLING_RATE = 256  # Sampling rate for CHB-MIT dataset
WINDOW_SAMPLES = WINDOW_SIZE * SAMPLING_RATE  # Number of samples per segment

EDF_DIR = '/content/chbmit'  # Path to EDF files
ANNOTATION_FILE = 'chb-summary.csv'  # Seizure summary file (default)
```

Function for extracting segments from files

In [12]:
```
def extract_segments(edf_file, seizure_starts, seizure_ends):
    raw = mne.io.read_raw_edf(edf_file, preload=True, verbose=False)
    data = raw.get_data()
    num_samples = data.shape[1]
    num_channels = data.shape[0]

    X = []
    y = []

    # Sliding over the entire file with a fixed-size window
    for start in range(0, num_samples - WINDOW_SAMPLES, WINDOW_SAMPLES):
        end = start + WINDOW_SAMPLES
        segment = data[:, start:end]

        label = 0  # Default: no seizure
        for sz_start, sz_end in zip(seizure_starts, seizure_ends):
            sz_start_sample = int(sz_start * SAMPLING_RATE)
            sz_end_sample = int(sz_end * SAMPLING_RATE)
            if end > sz_start_sample - 256 and start < sz_start_sample:
                label = 1  # Segment just before seizure onset
                break

        X.append(segment)
        y.append(label)

    return np.array(X), np.array(y)
```

General function for reading files and creating dataset

In [13]:
```
def process_all_patients(base_dir, summary_csv):
    summary_df = pd.read_csv(summary_csv)
    all_X = []
    all_y = []

    for _, row in tqdm(summary_df.iterrows(), total=len(summary_df)):
        patient = row['patient']
        filename = row['filename']
        seizures = eval(row['seizure_intervals'])

        seizure_starts = [s[0] for s in seizures]
        seizure_ends = [s[1] for s in seizures]

        full_path = os.path.join(base_dir, patient, filename)
        if not os.path.exists(full_path):
            continue

        X, y = extract_segments(full_path, seizure_starts, seizure_ends)
        all_X.append(X)
        all_y.append(y)

    X = np.concatenate(all_X, axis=0)
    y = np.concatenate(all_y, axis=0)
    return X, y
```

Creating csv summary file

In [14]:
```
import os
import pandas as pd
import re

SUMMARY_FILES = [
    "/content/drive/MyDrive/G_Pr/chb/chb01/chb01-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb02/chb02-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb03/chb03-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb04/chb04-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb05/chb05-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb06/chb06-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb07/chb07-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb08/chb08-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb09/chb09-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb10/chb10-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb11/chb11-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb12/chb12-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb13/chb13-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb14/chb14-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb15/chb15-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb16/chb16-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb17/chb17-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb18/chb18-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb19/chb19-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb20/chb20-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb21/chb21-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb22/chb22-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb23/chb23-summary.txt",
    "/content/drive/MyDrive/G_Pr/chb/chb24/chb24-summary.txt"

]

summary_data = []

for file_path in SUMMARY_FILES:
    with open(file_path, 'r') as file:
        lines = file.readlines()

    patient = os.path.basename(file_path).split('-')[0]
    current_edf = None
    seizure_times = []

    for line in lines:
```

```
        line = line.strip().lower()

        if "file name:" in line:
            current_edf = line.split(":")[1].strip()
            seizure_times = []

        elif "seizure start time" in line:
            # Extract number using regex
            start_time = int(re.search(r'\d+', line).group())
            seizure_times.append([start_time, -1])

        elif "seizure end time" in line and seizure_times:
            end_time = int(re.search(r'\d+', line).group())
            seizure_times[-1][1] = end_time

            summary_data.append({
                "patient": patient,
                "file": current_edf,
                "seizure_start": seizure_times[-1][0],
                "seizure_end": seizure_times[-1][1]
            })

df = pd.DataFrame(summary_data)
df.to_csv("chb-summary.csv", index=False)
print(" Summary file created: chb-summary.csv")
```

```
 Summary file created: chb-summary.csv
```

In [15]:
```
# +++
import os
import pandas as pd
import numpy as np
import mne
from sklearn.utils import shuffle
from joblib import dump

EDF_DIR = "/content/drive/MyDrive/G_Pr/chb"
SUMMARY_FILE = "chb-summary.csv"
WINDOW_SIZE = 2560  # 10 seconds at 256 Hz
STEP_SIZE = 2560    # non-overlapping windows

def extract_features_from_edf(edf_path, seizure_intervals):
    raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)

    raw_data = raw.get_data()
    sfreq = int(raw.info['sfreq'])  # typically 256 Hz

    seizure_data = []
    non_seizure_data = []

    n_channels, n_samples = raw_data.shape

    for start in range(0, n_samples - WINDOW_SIZE, STEP_SIZE):
        end = start + WINDOW_SIZE
        window = raw_data[:, start:end]

        window_start_sec = start // sfreq
        window_end_sec = end // sfreq

        label = 0  # non-seizure by default
        for s_start, s_end in seizure_intervals:
            if (window_start_sec < s_end and window_end_sec > s_start):
                label = 1
                break

        if label == 1:
            seizure_data.append(window)
        else:
            non_seizure_data.append(window)

    # Balance classes
```

```
    min_len = min(len(seizure_data), len(non_seizure_data))
    seizure_data = seizure_data[:min_len]
    non_seizure_data = non_seizure_data[:min_len]

    X = np.array(seizure_data + non_seizure_data)
    y = np.array([1]*len(seizure_data) + [0]*len(non_seizure_data))

    return X, y

def build_dataset_from_summary(summary_file, edf_dir):
    df = pd.read_csv(summary_file)
    all_X = []
    all_y = []

    expected_shape = None

    for i, row in df.iterrows():
        edf_path = os.path.join(edf_dir, row["patient"], row["file"])
        if not os.path.exists(edf_path):
            print(f"File not found: {edf_path}")
            continue

        if row["seizure_start"] == -1:
            seizure_intervals = []
        else:
            seizure_intervals = [(int(row["seizure_start"]), int(row["seizure_end"]))]

        try:
            X, y = extract_features_from_edf(edf_path, seizure_intervals)
        except Exception as e:
            print(f"Error processing {edf_path}: {e}")
            continue

        # Only Keep files with same channels
        if expected_shape is None:
            expected_shape = X.shape[1:]
        elif X.shape[1:] != expected_shape:
            print(f"Skipping due to shape mismatch: {edf_path}")
            continue

        all_X.append(X)
        all_y.append(y)
```

```
    X = np.concatenate(all_X, axis=0)
    y = np.concatenate(all_y, axis=0)
    X, y = shuffle(X, y, random_state=42)

    return X, y

# Run and save
X, y = build_dataset_from_summary(SUMMARY_FILE, EDF_DIR)
print("Final dataset shape:", X.shape, y.shape)

# Save using joblib
dump((X, y), "eeg_dataset.joblib")
print("Dataset saved as eeg_dataset.joblib")
```

```
<ipython-input-15-3d8509edccf9>:15: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8
-P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)
<ipython-input-15-3d8509edccf9>:15: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8
-P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)
<ipython-input-15-3d8509edccf9>:15: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8
-P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)
<ipython-input-15-3d8509edccf9>:15: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8
-P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)
<ipython-input-15-3d8509edccf9>:15: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8
-P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)
<ipython-input-15-3d8509edccf9>:15: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8
-P8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf(edf_path, preload=True, verbose=False)
<ipython-input-15-3d8509edccf9>:15: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8
```

In [16]:
```
import joblib
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle

# Load the dataset
X, y = joblib.load('/content/eeg_dataset.joblib')

# Ensure the data type is float32 for compatibility with Keras models
X = X.astype(np.float32)
y = y.astype(np.int32)

# Shuffle the data
X, y = shuffle(X, y, random_state=42)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

```
Train shape: (395, 23, 2560)
Test shape: (99, 23, 2560)
```

CNN Model

In [17]:
```
print("Train class distribution:", np.bincount(y_train))
print("Test class distribution:", np.bincount(y_test))
```

```
Train class distribution: [197 198]
Test class distribution: [50 49]
```

In [18]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout, BatchNormalization, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.regularizers import l2
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

# === Reshape EEG input to [samples, time, channels] ===
X_train_reshaped = X_train.reshape((X_train.shape[0], X_train.shape[2], X_train.shape[1]))
X_test_reshaped = X_test.reshape((X_test.shape[0], X_test.shape[2], X_test.shape[1]))

# === Normalize using standardization ===
mean = np.mean(X_train_reshaped, axis=(0, 1), keepdims=True)
std = np.std(X_train_reshaped, axis=(0, 1), keepdims=True)
X_train_norm = (X_train_reshaped - mean) / (std + 1e-7)
X_test_norm = (X_test_reshaped - mean) / (std + 1e-7)

# === Build optimized 1D-CNN model (simplified & regularized) ===
model = Sequential([
    Input(shape=(X_train_norm.shape[1], X_train_norm.shape[2])),

    Conv1D(16, kernel_size=7, activation='relu', kernel_regularizer=l2(0.002)),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),

    Conv1D(32, kernel_size=5, activation='relu', kernel_regularizer=l2(0.002)),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),

    Conv1D(64, kernel_size=3, activation='relu', kernel_regularizer=l2(0.002)),
    BatchNormalization(),
    MaxPooling1D(pool_size=2),

    Flatten(),
    Dropout(0.6),

    Dense(64, activation='relu', kernel_regularizer=l2(0.002)),
    BatchNormalization(),
    Dropout(0.5),

    Dense(1, activation='sigmoid')
```

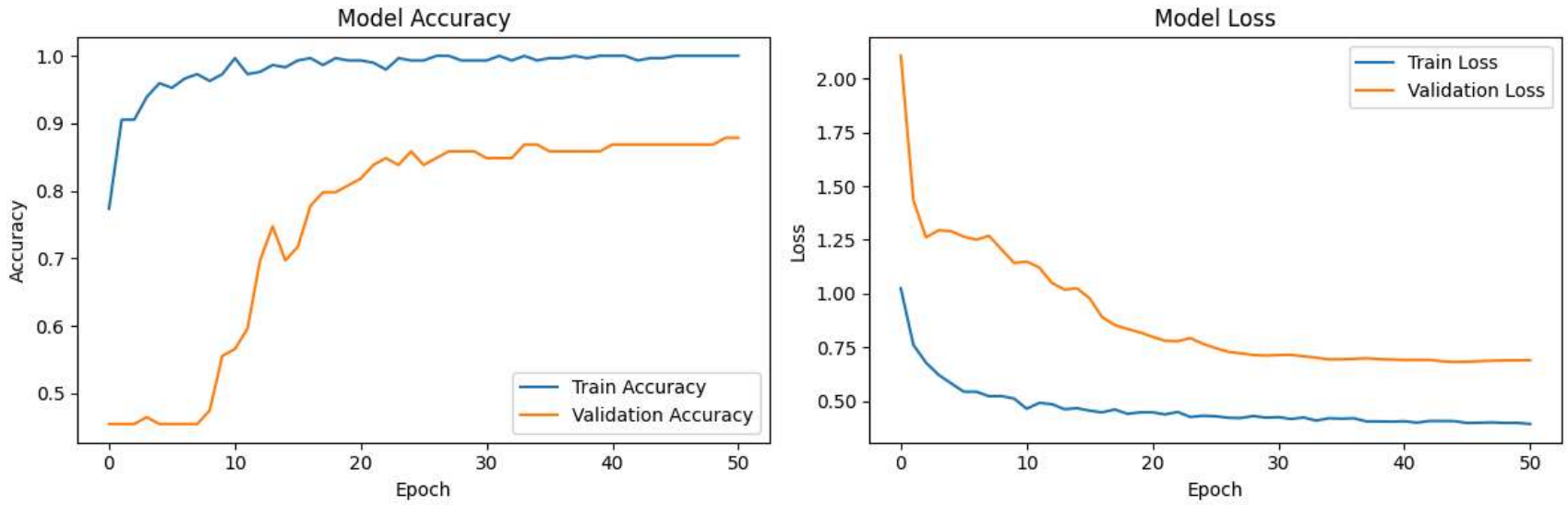
```
])

# === Compile the model ===
model.compile(optimizer=Adam(learning_rate=0.0003),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# === Callbacks for improved training ===
early_stop = EarlyStopping(monitor='val_loss', patience=6, restore_best_weights=True, verbose=1)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6, verbose=1)

# Save the best model during training
checkpoint = ModelCheckpoint("best_model.keras", monitor='val_loss',
                             save_best_only=True, mode='min', verbose=1)

# === Train the model ===
history = model.fit(X_train_norm, y_train,
                    epochs=100,
                    batch_size=32,
                    validation_split=0.25,
                    verbose=1,
                    callbacks=[early_stop, reduce_lr, checkpoint])

# === Load best weights if needed (optional after training) ===
model.load_weights("best_model.keras")
```

```
Epoch 1/100
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 435ms/step - accuracy: 0.7261 - loss: 1.0992
Epoch 1: val_loss improved from inf to 2.10646, saving model to best_model.keras
10/10 ━━━━━━━━━━━━━━━━━━━━ 13s 611ms/step - accuracy: 0.7304 - loss: 1.0924 - val_accuracy: 0.4545 - val_los
s: 2.1065 - learning_rate: 3.0000e-04
Epoch 2/100
 7/10 ━━━━━━━━━━━━━━──────  0s 9ms/step - accuracy: 0.8668 - loss: 0.9453
Epoch 2: val_loss improved from 2.10646 to 1.43377, saving model to best_model.keras
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - accuracy: 0.8799 - loss: 0.8808 - val_accuracy: 0.4545 - val_loss:
1.4338 - learning_rate: 3.0000e-04
Epoch 3/100
 7/10 ━━━━━━━━━━━━━━──────  0s 8ms/step - accuracy: 0.9040 - loss: 0.6624
Epoch 3: val_loss improved from 1.43377 to 1.26167, saving model to best_model.keras
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.9036 - loss: 0.6699 - val_accuracy: 0.4545 - val_loss:
1.2617 - learning_rate: 3.0000e-04
Epoch 4/100
 8/10 ━━━━━━━━━━━━━━━━────  0s 8ms/step - accuracy: 0.9453 - loss: 0.6194
Epoch 4: val_loss did not improve from 1.26167
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.9435 - loss: 0.6206 - val_accuracy: 0.4646 - val_loss:
```

In [19]:
```
# === Predict probabilities (sigmoid output) ===
y_pred = model.predict(X_test_norm).ravel()  # Probabilities (values between 0 and 1)

# === Convert probabilities to binary class predictions ===
y_pred_binary = (y_pred >= 0.5).astype(int)

# === Evaluation Metrics ===
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred_binary))
print("Accuracy Score:", accuracy_score(y_test, y_pred_binary))
print("ROC AUC Score:", roc_auc_score(y_test, y_pred))  # Not y_pred_binary
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 149ms/step

Classification Report:

              precision    recall  f1-score   support

           0       0.90      0.92      0.91        50
           1       0.92      0.90      0.91        49

    accuracy                           0.91        99
   macro avg       0.91      0.91      0.91        99
weighted avg       0.91      0.91      0.91        99

Accuracy Score: 0.9090909090909091
ROC AUC Score: 0.9510204081632654
```

In [20]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

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



In [21]:
```
!pip install mne --quiet
!pip install matplotlib numpy --quiet
```

In [22]:
```
import mne
import numpy as np
import matplotlib.pyplot as plt
```

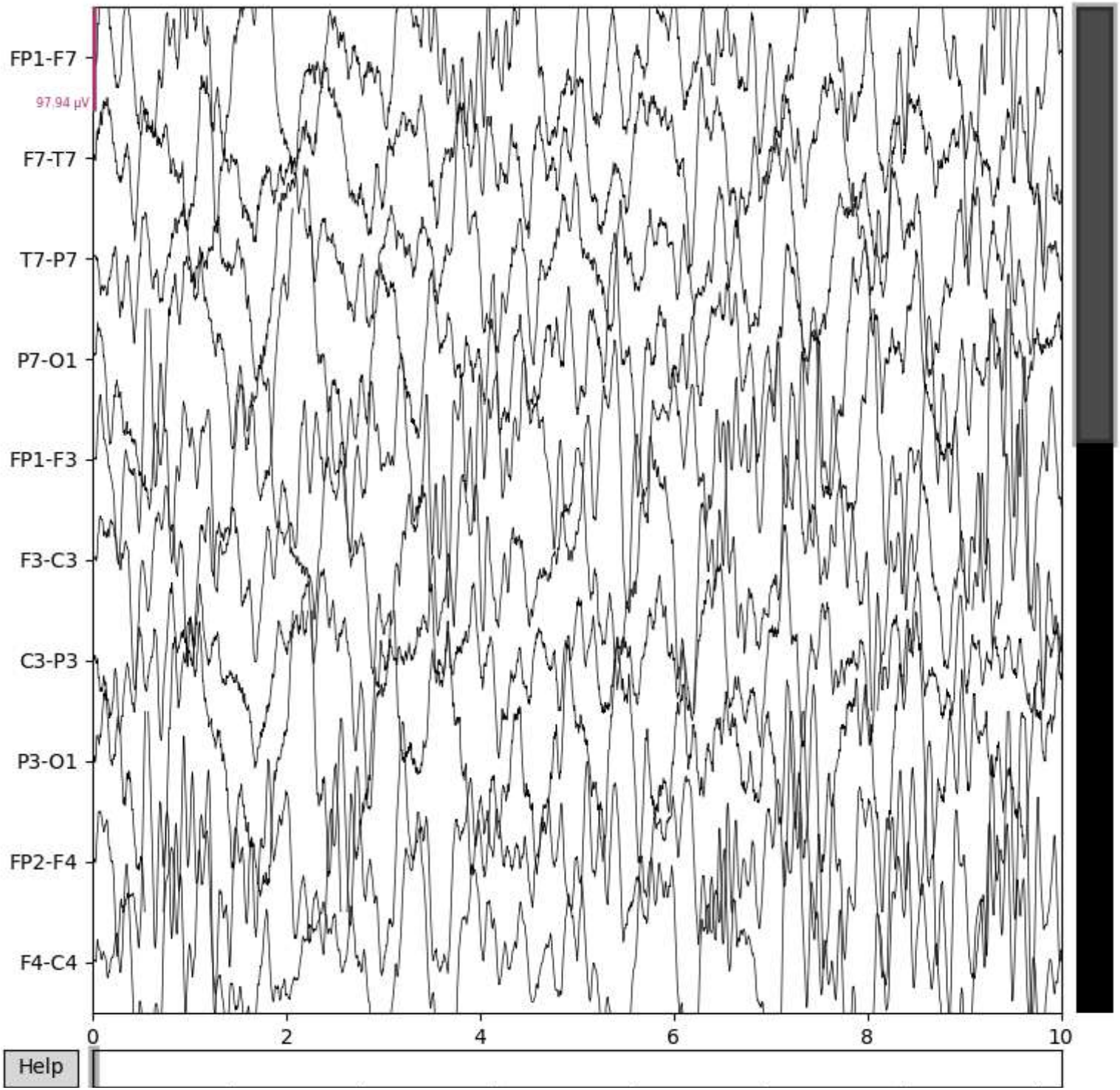
In [23]:
```
# Mount Google Drive to access EEG data
from google.colab import drive
drive.mount('/content/drive')

# Check the contents of the chb directory
import os
```

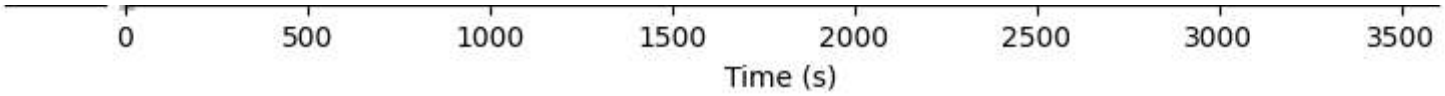
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", fo
rce_remount=True).
```

In [24]:
```
# Path to the EEG file with a seizure
seizure_file = "/content/drive/MyDrive/G_Pr/chb/chb01/chb01_15.edf"

# Path to the normal EEG file
normal_file = "/content/drive/MyDrive/G_Pr/chb/chb01/chb01_09.edf"

# Load the EEG file with a seizure
raw_seizure = mne.io.read_raw_edf(seizure_file, preload=True)

# Load the normal EEG file
raw_normal = mne.io.read_raw_edf(normal_file, preload=True)
```

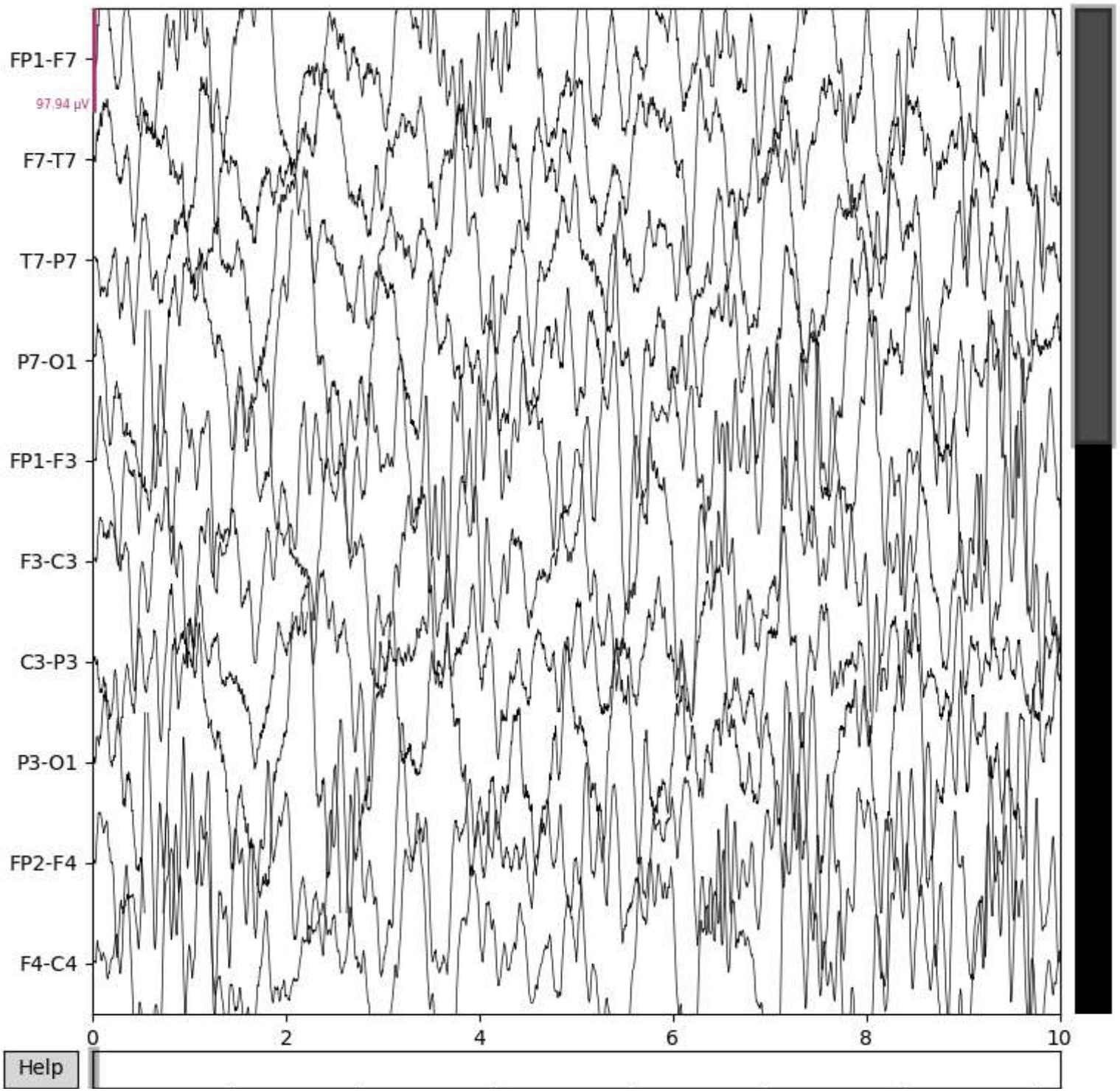
```
Extracting EDF parameters from /content/drive/MyDrive/G_Pr/chb/chb01/chb01_15.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 921599  =      0.000 ...  3599.996 secs...

<ipython-input-24-a1badd0fd9ad>:8: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8-P
8'}. Applying running numbers for duplicates.
  raw_seizure = mne.io.read_raw_edf(seizure_file, preload=True)

Extracting EDF parameters from /content/drive/MyDrive/G_Pr/chb/chb01/chb01_09.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 921599  =      0.000 ...  3599.996 secs...

<ipython-input-24-a1badd0fd9ad>:11: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8-P
8'}. Applying running numbers for duplicates.
  raw_normal = mne.io.read_raw_edf(normal_file, preload=True)
```

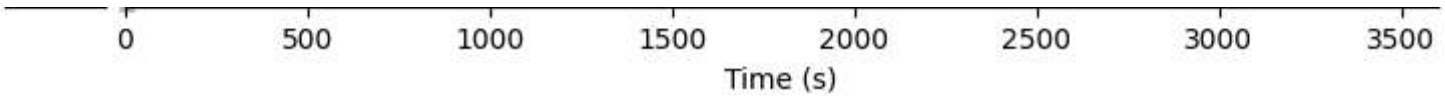
In [25]:
```
# Display information about the seizure EEG file
raw_seizure.info

# Plot the first few seconds of the seizure EEG file for inspection
raw_seizure.plot(n_channels=10, duration=10, scalings='auto')
```


FP1-F7
97.94 µV
F7-T7
T7-P7
P7-O1
FP1-F3
F3-C3
C3-P3
P3-O1
FP2-F4
F4-C4
0
2
4
6
8
10
Help


0
500
1000
1500
2000
2500
3000
3500
Time (s)

Out[25]:


FP1-F7
97.94 µV
F7-T7
T7-P7
P7-O1
FP1-F3
F3-C3
C3-P3
P3-O1
FP2-F4
F4-C4
0
2
4
6
8
10
Help



In [26]:
```python
def segment_raw_data(raw, segment_duration_sec=5):
    """
    Segments raw EEG data into fixed-duration windows.

    Parameters:
        raw (mne.io.Raw): Raw EEG data
        segment_duration_sec (int): Duration of each segment in seconds

    Returns:
        List of numpy arrays, each representing a segment
    """
    sampling_rate = int(raw.info['sfreq'])
    segment_samples = segment_duration_sec * sampling_rate
    data, _ = raw[:]
    num_segments = data.shape[1] // segment_samples
    segments = []

    for i in range(num_segments):
        start = i * segment_samples
        end = start + segment_samples
        segment = data[:, start:end]
        if segment.shape[1] == segment_samples:
            segments.append(segment)

    return segments
```

In [27]:
```python
# Segment both seizure and normal data into 5-second chunks
seizure_segments = segment_raw_data(raw_seizure, segment_duration_sec=5)
normal_segments = segment_raw_data(raw_normal, segment_duration_sec=5)

print(f"Number of seizure segments: {len(seizure_segments)}")
print(f"Number of normal segments: {len(normal_segments)}")
```

```
Number of seizure segments: 720
Number of normal segments: 720
```

In [28]:
```
import mne

# Load the EDF file (you can change to your own path)
raw = mne.io.read_raw_edf("/content/drive/MyDrive/G_Pr/chb/chb01/chb01_15.edf", preload=True)

# Print the list of EEG channel names
print("Channel names in this EDF file:")
print(raw.ch_names)
```

```
Extracting EDF parameters from /content/drive/MyDrive/G_Pr/chb/chb01/chb01_15.edf...
EDF file detected
Setting channel info structure...
Creating raw.info structure...
Reading 0 ... 921599  =      0.000 ...  3599.996 secs...

<ipython-input-28-1ed393ed5fb4>:4: RuntimeWarning: Channel names are not unique, found duplicates for: {'T8-P
8'}. Applying running numbers for duplicates.
  raw = mne.io.read_raw_edf("/content/drive/MyDrive/G_Pr/chb/chb01/chb01_15.edf", preload=True)

Channel names in this EDF file:
['FP1-F7', 'F7-T7', 'T7-P7', 'P7-O1', 'FP1-F3', 'F3-C3', 'C3-P3', 'P3-O1', 'FP2-F4', 'F4-C4', 'C4-P4', 'P4-O
2', 'FP2-F8', 'F8-T8', 'T8-P8-0', 'P8-O2', 'FZ-CZ', 'CZ-PZ', 'P7-T7', 'T7-FT9', 'FT9-FT10', 'FT10-T8', 'T8-P8-
1']
```

Creating Spectogram image from segments

In [29]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import spectrogram
from tqdm import tqdm

def save_multi_channel_spectrograms(segments, raw_info, label, output_dir):
    selected_channels = ['FP1-F3', 'F7-T7', 'T7-P7']
    channel_indices = [raw_info['ch_names'].index(ch) for ch in selected_channels if ch in raw_info['ch_names']]

    label_dir = os.path.join(output_dir, label)
    os.makedirs(label_dir, exist_ok=True)

    for i, segment in tqdm(enumerate(segments), total=len(segments)):
        selected_signals = [segment[idx] for idx in channel_indices]
        combined_signal = np.mean(selected_signals, axis=0)

        f, t, Sxx = spectrogram(combined_signal, fs=256)

        plt.figure(figsize=(2.24, 2.24))
        plt.pcolormesh(t, f, 10 * np.log10(Sxx), shading='gouraud')
        plt.axis('off')
        plt.tight_layout(pad=0)
        filename = os.path.join(label_dir, f'{label}_{i}.png')
        plt.savefig(filename, dpi=100, bbox_inches='tight', pad_inches=0)
        plt.close()
```

In [30]:
```
save_multi_channel_spectrograms(seizure_segments, raw.info, label='seizure', output_dir='/content/eeg_spectrogra
save_multi_channel_spectrograms(normal_segments, raw.info, label='normal', output_dir='/content/eeg_spectrograms
```

```
100%|██████████| 720/720 [00:34<00:00, 21.02it/s]
100%|██████████| 720/720 [00:35<00:00, 20.53it/s]
```

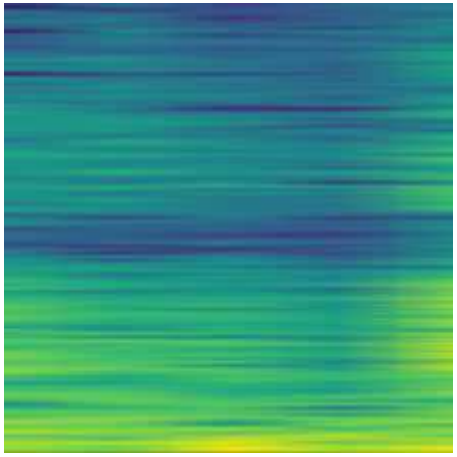
In [31]:
```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.11/dist-packages (4.11.0.86)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.11/dist-packages (from opencv-python)
(2.0.2)
```

In [32]:
```
import cv2
import os
from natsort import natsorted  # For numerical sorting of files

# Path to the folder where spectrogram images are stored
image_folder = '/content/eeg_spectrograms/normal/'

# Output video file name
video_name = 'eeg_spectrogram_video.avi'

# Get the list of images and sort them numerically
images = natsorted([img for img in os.listdir(image_folder) if img.endswith(".png")])

# Read the dimensions of the first image to set the video size
first_frame = cv2.imread(os.path.join(image_folder, images[0]))
height, width, _ = first_frame.shape

# Define the output video using OpenCV
video_writer = cv2.VideoWriter(video_name, cv2.VideoWriter_fourcc(*'XVID'), 5, (width, height))

# Add images to the video
for image in images:
    img_path = os.path.join(image_folder, image)
    frame = cv2.imread(img_path)
    video_writer.write(frame)

video_writer.release()
print(" Video successfully created:", video_name)
```

```
 Video successfully created: eeg_spectrogram_video.avi
```

In [33]:
```
from google.colab.patches import cv2_imshow
import cv2

cap = cv2.VideoCapture('eeg_spectrogram_video.avi')

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    cv2_imshow(frame)
    # Pause to view the frame - waitKey doesn't work in Colab, so we use sleep instead
    import time
    time.sleep(0.5)  # 0.5 seconds between each frame

cap.release()
```



In [34]:
```
# Convert video from AVI to MP4 using ffmpeg
!ffmpeg -i eeg_spectrogram_video.avi -vcodec libx264 eeg_spectrogram_video.mp4
```

```
ffmpeg version 4.4.2-0ubuntu0.22.04.1 Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 11 (Ubuntu 11.2.0-19ubuntu1)
  configuration: --prefix=/usr --extra-version=0ubuntu0.22.04.1 --toolchain=hardened --libdir=/usr/lib/x86_64-
linux-gnu --incdir=/usr/include/x86_64-linux-gnu --arch=amd64 --enable-gpl --disable-stripping --enable-gnutls
--enable-ladspa --enable-libaom --enable-libass --enable-libbluray --enable-libbs2b --enable-libcaca --enable-
libcdio --enable-libcodec2 --enable-libdav1d --enable-libflite --enable-libfontconfig --enable-libfreetype --e
nable-libfribidi --enable-libgme --enable-libgsm --enable-libjack --enable-libmp3lame --enable-libmysofa --ena
ble-libopenjpeg --enable-libopenmpt --enable-libopus --enable-libpulse --enable-librabbitmq --enable-librubber
band --enable-libshine --enable-libsnappy --enable-libsoxr --enable-libspeex --enable-libsrt --enable-libssh -
-enable-libtheora --enable-libtwolame --enable-libvidstab --enable-libvorbis --enable-libvpx --enable-libwebp
--enable-libx265 --enable-libxml2 --enable-libxvid --enable-libzimg --enable-libzmq --enable-libzvbi --enable-
lv2 --enable-omx --enable-openal --enable-opencl --enable-opengl --enable-sdl2 --enable-pocketsphinx --enable-
librsvg --enable-libmfx --enable-libdc1394 --enable-libdrm --enable-libiec61883 --enable-chromaprint --enable-
frei0r --enable-libx264 --enable-shared
  libavutil      56. 70.100 / 56. 70.100
  libavcodec     58.134.100 / 58.134.100
  libavformat    58. 76.100 / 58. 76.100
  libavdevice    58. 13.100 / 58. 13.100
  libavfilter     7.110.100 /  7.110.100
  libswscale      5.  9.100 /  5.  9.100
  libswresample   3.  9.100 /  3.  9.100
  libpostproc    55.  9.100 / 55.  9.100
Input #0, avi, from 'eeg_spectrogram_video.avi':
  Metadata:
    software        : Lavf59.27.100
  Duration: 00:02:24.00, start: 0.000000, bitrate: 240 kb/s
  Stream #0:0: Video: mpeg4 (Simple Profile) (XVID / 0x44495658), yuv420p, 224x224 [SAR 1:1 DAR 1:1], 239 kb/
s, 5 fps, 5 tbr, 5 tbn, 5 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (mpeg4 (native) -> h264 (libx264))
Press [q] to stop, [?] for help
[libx264 @ 0x5b551b7bb580] using SAR=1/1
[libx264 @ 0x5b551b7bb580] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2 AVX512
[libx264 @ 0x5b551b7bb580] profile High, level 1.1, 4:2:0, 8-bit
[libx264 @ 0x5b551b7bb580] 264 - core 163 r3060 5db6aa6 - H.264/MPEG-4 AVC codec - Copyleft 2003-2021 - htt
p://www.videolan.org/x264.html (http://www.videolan.org/x264.html) - options: cabac=1 ref=3 deblock=1:0:0 anal
yse=0x3:0x113 me=hex subme=7 psy=1 psy_rd=1.00:0.00 mixed_ref=1 me_range=16 chroma_me=1 trellis=1 8x8dct=1 cqm
=0 deadzone=21,11 fast_pskip=1 chroma_qp_offset=-2 threads=7 lookahead_threads=1 sliced_threads=0 nr=0 decimat
e=1 interlaced=0 bluray_compat=0 constrained_intra=0 bframes=3 b_pyramid=2 b_adapt=1 b_bias=0 direct=1 weightb
=1 open_gop=0 weightp=2 keyint=250 keyint_min=5 scenecut=40 intra_refresh=0 rc_lookahead=40 rc=crf mbtree=1 cr
f=23.0 qcomp=0.60 qpmin=0 qpmax=69 qpstep=4 ip_ratio=1.40 aq=1:1.00
Output #0, mp4, to 'eeg_spectrogram_video.mp4':
  Metadata:
```

```
    software        : Lavf59.27.100
    encoder         : Lavf58.76.100
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv420p(progressive), 224x224 [SAR 1:1 DAR 1:1], q=2-31, 5 fp
s, 10240 tbn
    Metadata:
      encoder         : Lavc58.134.100 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
frame=  720 fps=402 q=-1.0 Lsize=    1989kB time=00:02:23.40 bitrate= 113.6kbits/s speed=  80x
video:1985kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 0.186364%
[libx264 @ 0x5b551b7bb580] frame I:3     Avg QP:19.81  size:  3485
[libx264 @ 0x5b551b7bb580] frame P:717   Avg QP:21.89  size:  2819
[libx264 @ 0x5b551b7bb580] mb I  I16..4:  1.7% 94.9%  3.4%
[libx264 @ 0x5b551b7bb580] mb P  I16..4:  4.5% 78.2% 11.7%  P16..4:  2.2%  3.0%  0.5%  0.0%  0.0%    skip: 0.
0%
[libx264 @ 0x5b551b7bb580] 8x8 transform intra:83.0% inter:98.9%
[libx264 @ 0x5b551b7bb580] coded y,uvDC,uvAC intra: 91.2% 99.8% 96.7% inter: 94.1% 98.5% 80.0%
[libx264 @ 0x5b551b7bb580] i16 v,h,dc,p:  0% 99%  0%  1%
[libx264 @ 0x5b551b7bb580] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu:  1% 94%  4%  0%  0%  0%  0%  0%  0%
[libx264 @ 0x5b551b7bb580] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu:  0% 98%  1%  0%  0%  0%  0%  0%  0%
[libx264 @ 0x5b551b7bb580] i8c dc,h,v,p:  9% 87%  1%  3%
[libx264 @ 0x5b551b7bb580] Weighted P-Frames: Y:0.3% UV:0.3%
[libx264 @ 0x5b551b7bb580] ref P L0: 38.5% 19.3% 22.8% 19.4%  0.0%
[libx264 @ 0x5b551b7bb580] kb/s:112.88
```

In [35]:
```
from IPython.display import HTML
from base64 import b64encode

mp4_file = "eeg_spectrogram_video.mp4"

# Read the video and convert it to HTML for playback inside Colab
mp4 = open(mp4_file, 'rb').read()
data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
HTML(f"""
<video width=600 controls>
    <source src="{data_url}" type="video/mp4">
</video>
""")
```

Out[35]:

Creating Dataset and Training CNN with PyTorch

In [36]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms, models
from PIL import Image
import os
```

Definition of Customized EEG dataset

In [37]:
```
class EEGDataset(Dataset):
    def __init__(self, image_dir, label, transform=None):
        self.image_dir = image_dir
        self.label = label
        self.transform = transform
        self.image_paths = [os.path.join(image_dir, fname) for fname in os.listdir(image_dir) if fname.endswith(

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image = Image.open(self.image_paths[idx]).convert('RGB')
        if self.transform:
            image = self.transform(image)
        return image, self.label
```

In [38]:
```
# Transformations (resize to 224x224 and normalize for ResNet)
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5]*3, std=[0.5]*3)
])

# Create datasets
seizure_dataset = EEGDataset('/content/eeg_spectrograms/seizure', label=1, transform=transform)
normal_dataset = EEGDataset('/content/eeg_spectrograms/normal', label=0, transform=transform)

# Combine and split
full_dataset = seizure_dataset + normal_dataset
train_size = int(0.8 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(full_dataset, [train_size, val_size])

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
```

Use from Pretrained ResNet18 dataset

In [39]:
```
import torch.nn as nn
from torchvision import models

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

base_model = models.resnet18(pretrained=True)

# Freeze all layers except the last few (Optional)
for param in base_model.parameters():
    param.requires_grad = False

# Replace the fully connected layer with one including dropout
base_model.fc = nn.Sequential(
    nn.Dropout(p=0.5),              # Dropout 50%
    nn.Linear(base_model.fc.in_features, 2)
)

model = base_model.to(device)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrain
ed' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in the future. The current be
havior is equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet18_W
eights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoint
s/resnet18-f37072fd.pth
100%|██████████| 44.7M/44.7M [00:00<00:00, 228MB/s]
```

In [40]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

In [41]:
```
import copy
import torch
import torch.nn as nn

num_epochs = 50  # Increase the number of epochs since we have EarlyStopping

# Set up Early Stopping
patience = 5
best_val_loss = float('inf')
best_model_wts = copy.deepcopy(model.state_dict())
epochs_no_improve = 0

# Add weight decay to optimizer (L2 Regularization)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4)

# Scheduler to reduce LR when validation loss stops improving
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=2, factor=0.5, verbose=Tr

train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(num_epochs):
    # ----------- Training ----------
    model.train()
    total_train_loss = 0
    correct_train = 0
    total_train = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        total_train_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    avg_train_loss = total_train_loss / len(train_loader)
```

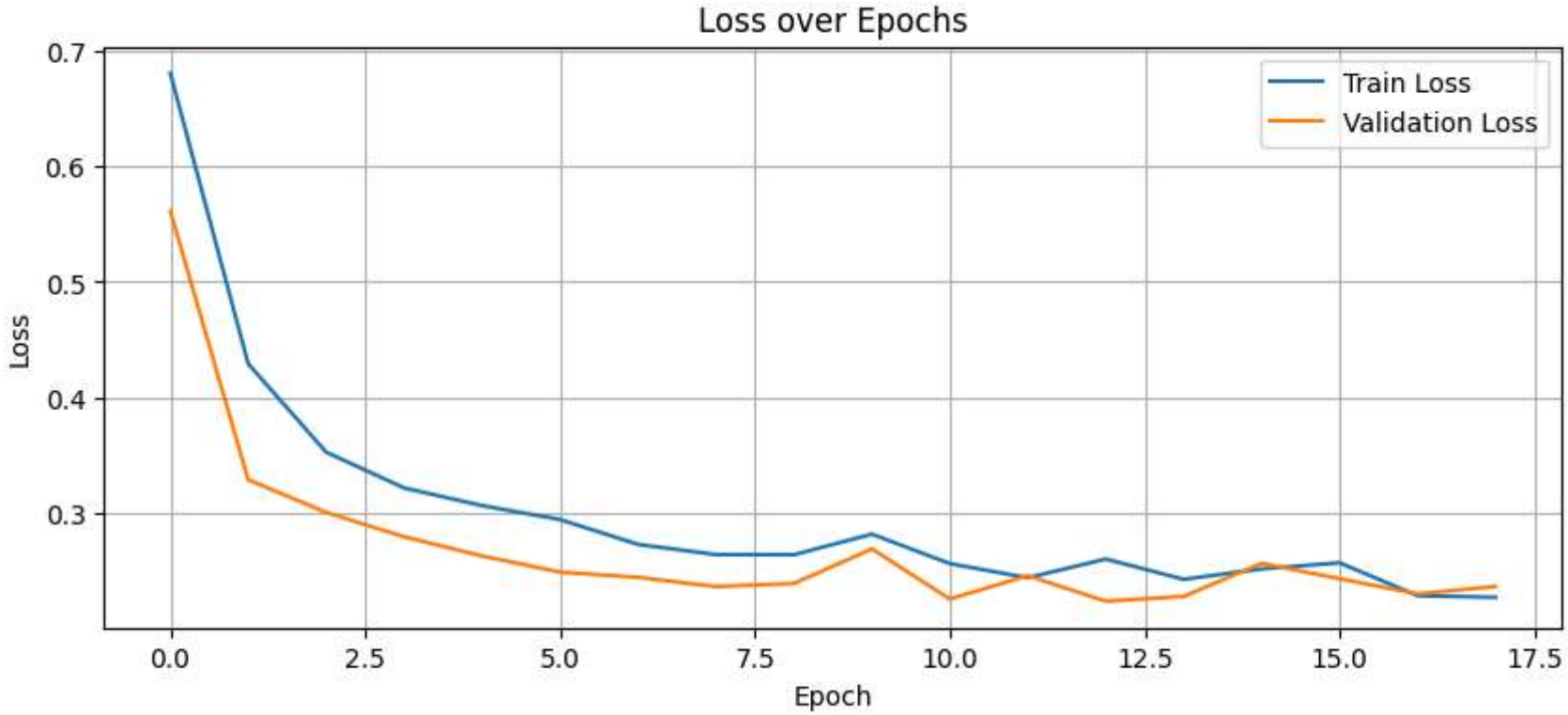
```
train_accuracy = 100 * correct_train / total_train
train_losses.append(avg_train_loss)
train_accuracies.append(train_accuracy)

# ---------- Validation ----------
model.eval()
total_val_loss = 0
correct_val = 0
total_val = 0

with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)

        total_val_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total_val += labels.size(0)
        correct_val += (predicted == labels).sum().item()

avg_val_loss = total_val_loss / len(val_loader)
val_accuracy = 100 * correct_val / total_val
val_losses.append(avg_val_loss)
val_accuracies.append(val_accuracy)

scheduler.step(avg_val_loss)

print(f"Epoch [{epoch+1}/{num_epochs}] "
      f"Train Loss: {avg_train_loss:.4f}, Train Acc: {train_accuracy:.2f}% "
      f"| Val Loss: {avg_val_loss:.4f}, Val Acc: {val_accuracy:.2f}%")

# -------- Early Stopping --------
if avg_val_loss < best_val_loss:
    best_val_loss = avg_val_loss
    best_model_wts = copy.deepcopy(model.state_dict())
    epochs_no_improve = 0
else:
    epochs_no_improve += 1
    if epochs_no_improve >= patience:
        print(" Early stopping activated.")
        break
```

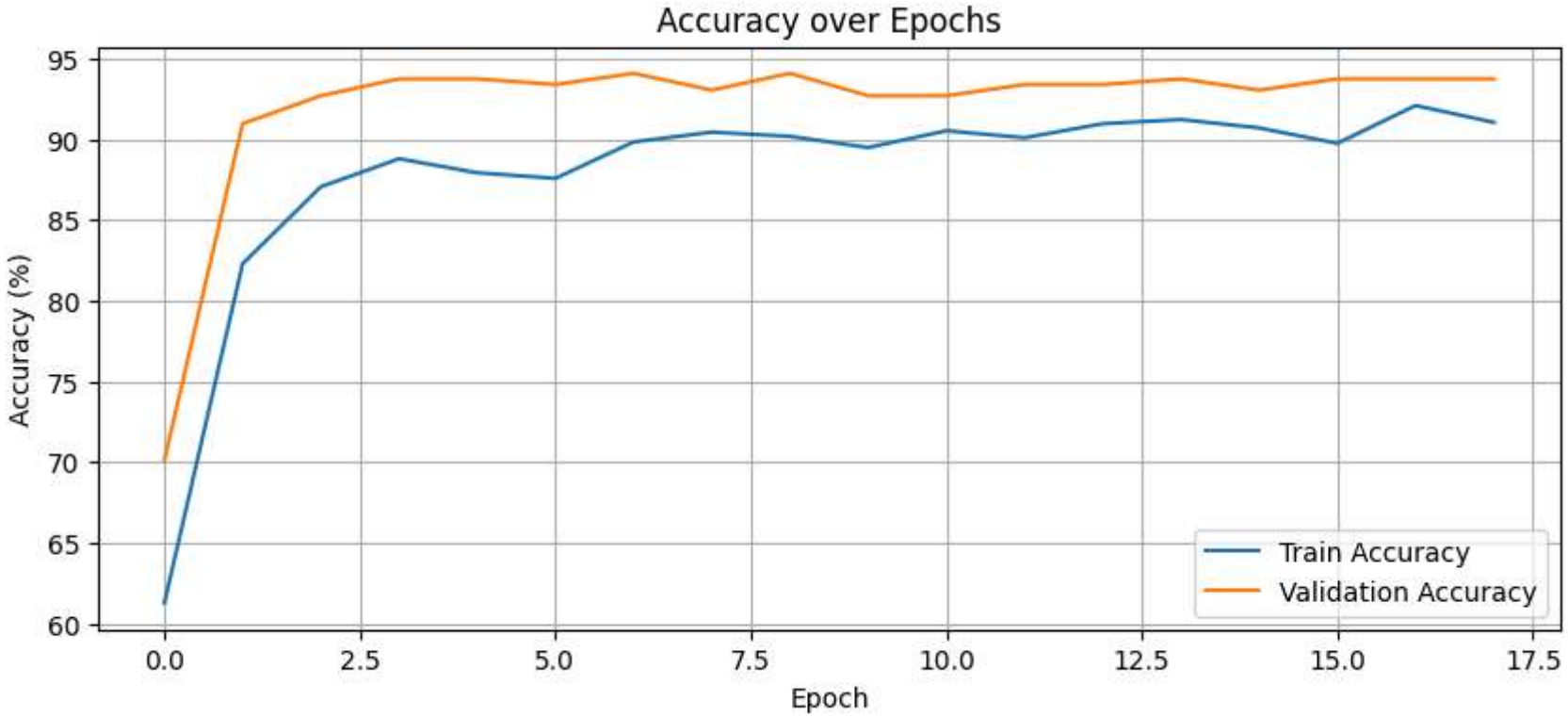
```
# Load the best weights
model.load_state_dict(best_model_wts)
```

```
/usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is
deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn(

Epoch [1/50] Train Loss: 0.6804, Train Acc: 61.28% | Val Loss: 0.5614, Val Acc: 70.14%
Epoch [2/50] Train Loss: 0.4293, Train Acc: 82.29% | Val Loss: 0.3291, Val Acc: 90.97%
Epoch [3/50] Train Loss: 0.3529, Train Acc: 87.07% | Val Loss: 0.3009, Val Acc: 92.71%
Epoch [4/50] Train Loss: 0.3220, Train Acc: 88.80% | Val Loss: 0.2799, Val Acc: 93.75%
Epoch [5/50] Train Loss: 0.3068, Train Acc: 87.93% | Val Loss: 0.2633, Val Acc: 93.75%
Epoch [6/50] Train Loss: 0.2948, Train Acc: 87.59% | Val Loss: 0.2494, Val Acc: 93.40%
Epoch [7/50] Train Loss: 0.2733, Train Acc: 89.84% | Val Loss: 0.2448, Val Acc: 94.10%
Epoch [8/50] Train Loss: 0.2645, Train Acc: 90.45% | Val Loss: 0.2369, Val Acc: 93.06%
Epoch [9/50] Train Loss: 0.2644, Train Acc: 90.19% | Val Loss: 0.2395, Val Acc: 94.10%
Epoch [10/50] Train Loss: 0.2821, Train Acc: 89.50% | Val Loss: 0.2694, Val Acc: 92.71%
Epoch [11/50] Train Loss: 0.2567, Train Acc: 90.54% | Val Loss: 0.2261, Val Acc: 92.71%
Epoch [12/50] Train Loss: 0.2445, Train Acc: 90.10% | Val Loss: 0.2463, Val Acc: 93.40%
Epoch [13/50] Train Loss: 0.2607, Train Acc: 90.97% | Val Loss: 0.2242, Val Acc: 93.40%
Epoch [14/50] Train Loss: 0.2431, Train Acc: 91.23% | Val Loss: 0.2284, Val Acc: 93.75%
Epoch [15/50] Train Loss: 0.2522, Train Acc: 90.71% | Val Loss: 0.2570, Val Acc: 93.06%
Epoch [16/50] Train Loss: 0.2574, Train Acc: 89.76% | Val Loss: 0.2436, Val Acc: 93.75%
Epoch [17/50] Train Loss: 0.2289, Train Acc: 92.10% | Val Loss: 0.2305, Val Acc: 93.75%
Epoch [18/50] Train Loss: 0.2276, Train Acc: 91.06% | Val Loss: 0.2370, Val Acc: 93.75%
 Early stopping activated.
```

Out[41]: <All keys matched successfully>

In [42]:
```
import matplotlib.pyplot as plt

# Loss
plt.figure(figsize=(10,4))
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Accuracy
plt.figure(figsize=(10,4))
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Validation Accuracy')
plt.title('Accuracy over Epochs')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True)
plt.show()
```


Loss over Epochs
Train Loss
Validation Loss
Loss
Epoch
0.7
0.6
0.5
0.4
0.3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Accuracy over Epochs
Accuracy (%)
Epoch
Train Accuracy
Validation Accuracy
95
90
85
80
75
70
65
60
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5