```
Heart Sound Classification using CNN & Transfer Learning
Group No-6
```

```
import os
import zipfile
import shutil
from pathlib import Path
import numpy as np
import json
from pathlib import Path

ZIP_PATH = Path("/content/archive (2).zip")


import matplotlib.pyplot as plt
plt.ioff()
```

```
<contextlib.ExitStack at 0x7b7e8db58f50>
```

AUDIO

```
from scipy import signal
try:
    import soundfile as sf
    def load_audio(path):
        x, sr = sf.read(str(path))
        if x.ndim > 1:
            x = x.mean(axis=1)
        return x.astype(np.float32), sr
except Exception:
    from scipy.io import wavfile
    def load_audio(path):
        sr, data = wavfile.read(str(path))
        if data.dtype.kind == 'i':
            data = data.astype(np.float32) / np.iinfo(data.dtype).max
        if data.ndim > 1:
            data = data.mean(axis=1)
        return data.astype(np.float32), sr
```

**Config **

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import (classification_report, confusion_matrix, accuracy_score,
                             precision_recall_curve, average_precision_score)
import sklearn
import tensorflow as tf
from tensorflow.keras import layers, models
```

```
ZIP_PATH = Path("/content/archive (2).zip")
EXTRACT_DIR = Path("/mnt/data/archive_extracted_2class")
OUTPUT_DIR = Path("/mnt/data")
TARGET_SR = 2000
MAX_SEC = 10.0
N_FFT = 256
HOP = 128
IMG_H = 128
IMG_W = 128
MAX_SAMPLES = 3000
BATCH_SIZE = 16
EPOCHS = 6            # raise to 20-50 for final runs
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)
```

Prepare extraction directory

```
ZIP_PATH = "/content/archive (2).zip"
EXTRACT_DIR = "/content/extracted"

!rm -rf "$EXTRACT_DIR"
!mkdir -p "$EXTRACT_DIR"
```

```
!unzip "$ZIP_PATH" -d "$EXTRACT_DIR"
print("Extracted dataset to:", EXTRACT_DIR)
```

```
  inflating: /content/extracted/validation/e00051.wav
  inflating: /content/extracted/validation/e00052.wav
  inflating: /content/extracted/validation/e00053.wav
  inflating: /content/extracted/validation/e00054.wav
  inflating: /content/extracted/validation/e00055.wav
  inflating: /content/extracted/validation/e00059.wav
  inflating: /content/extracted/validation/e00071.wav
  inflating: /content/extracted/validation/e00087.wav
  inflating: /content/extracted/validation/e00097.wav
  inflating: /content/extracted/validation/e00114.wav
  inflating: /content/extracted/validation/e00120.wav
  inflating: /content/extracted/validation/e00135.wav
  inflating: /content/extracted/validation/e00140.wav
  inflating: /content/extracted/validation/e00142.wav
  inflating: /content/extracted/validation/e00152.wav
  inflating: /content/extracted/validation/e00176.wav
  inflating: /content/extracted/validation/e00191.wav
  inflating: /content/extracted/validation/e00195.wav
  inflating: /content/extracted/validation/e00216.wav
  inflating: /content/extracted/validation/e00228.wav
  inflating: /content/extracted/validation/e00266.wav
  inflating: /content/extracted/validation/e00275.wav
  inflating: /content/extracted/validation/e00295.wav
  inflating: /content/extracted/validation/e00304.wav
  inflating: /content/extracted/validation/e00305.wav
  inflating: /content/extracted/validation/e00321.wav
  inflating: /content/extracted/validation/e00328.wav
  inflating: /content/extracted/validation/e00330.wav
  inflating: /content/extracted/validation/e00336.wav
  inflating: /content/extracted/validation/e00359.wav
  inflating: /content/extracted/validation/e00373.wav
  inflating: /content/extracted/validation/e00388.wav
  inflating: /content/extracted/validation/e00435.wav
  inflating: /content/extracted/validation/e00456.wav
  inflating: /content/extracted/validation/e00457.wav
  inflating: /content/extracted/validation/e00461.wav
  inflating: /content/extracted/validation/e00475.wav
  inflating: /content/extracted/validation/e00477.wav
  inflating: /content/extracted/validation/e00523.wav
  inflating: /content/extracted/validation/e00526.wav
  inflating: /content/extracted/validation/e00528.wav
  inflating: /content/extracted/validation/e00536.wav
  inflating: /content/extracted/validation/e00537.wav
  inflating: /content/extracted/validation/e00539.wav
  inflating: /content/extracted/validation/e00551.wav
  inflating: /content/extracted/validation/e00562.wav
  inflating: /content/extracted/validation/e00591.wav
  inflating: /content/extracted/validation/e00601.wav
  inflating: /content/extracted/validation/e00603.wav
  inflating: /content/extracted/validation/e00605.wav
  inflating: /content/extracted/validation/e00619.wav
  inflating: /content/extracted/validation/e00622.wav
  inflating: /content/extracted/validation/e00627.wav
  inflating: /content/extracted/validation/e00648.wav
  inflating: /content/extracted/validation/e00657.wav
  inflating: /content/extracted/validation/e00670.wav
  inflating: /content/extracted/validation/index.html
Extracted dataset to: /content/extracted
```

Find annotation CSVs

```
annotation_csvs = []
for root, dirs, files in os.walk(EXTRACT_DIR):
    for f in files:
        if f.lower().endswith('.csv') and ('reference' in f.lower() or 'reference_withsqi' in f.lower() or 'online_appendi
            annotation_csvs.append(Path(root) / f)
# fallback: any CSV
if len(annotation_csvs) == 0:
    for root, dirs, files in os.walk(EXTRACT_DIR):
        for f in files:
            if f.lower().endswith('.csv'):
                annotation_csvs.append(Path(root) / f)
annotation_csvs = sorted(set(annotation_csvs))
print("Annotation CSVs found:", annotation_csvs)
```

```
Annotation CSVs found: [PosixPath('/content/extracted/annotations/Online_Appendix_Diagnosis_meanings.csv'), PosixPath('/cont
```

Parse annotation CSVs -> mapping filename -> 'normal'/'abnormal'

```
mapping = {}
for csvp in annotation_csvs:
    try:
        df = pd.read_csv(csvp, dtype=str, keep_default_na=False)
    except Exception:
        try:
            df = pd.read_csv(csvp, encoding='latin1', dtype=str, keep_default_na=False)
        except Exception as e:
            print("Failed to read", csvp, ":", e)
            continue
    cols = [c.lower() for c in df.columns]
    # heuristics for file and label columns
    file_col = None
    for candidate in ['recording','recording_id','filename','file','audio','rec','record']:
        if candidate in cols:
            file_col = df.columns[cols.index(candidate)]; break
    if file_col is None:
        file_col = df.columns[0]
    label_col = None
    for candidate in ['label','diagnosis','diagnoses','target','annotation','reference','class','response']:
        if candidate in cols:
            label_col = df.columns[cols.index(candidate)]; break
    if label_col is None:
        for c in df.columns:
            vals = " ".join(df[c].astype(str).str.lower().unique().tolist())
            if 'normal' in vals or 'abnormal' in vals or 'poor' in vals or 'unknown' in vals:
                label_col = c; break
    if label_col is None:
        if len(df.columns) > 1:
            label_col = df.columns[1]
        else:
            continue

    for _, row in df.iterrows():
        fname = str(row[file_col]).strip()
        if fname == '' or fname.lower() in {'nan','na','none'}:
            continue
        fname_key = Path(fname).name
        labv = str(row[label_col]).strip().lower()
        mapped = None
        if 'normal' in labv or labv == '0' or labv == 'n':
            mapped = 'normal'
        if mapped is None and ('abnormal' in labv or 'murmur' in labv or labv == '1' or labv == 'a'):
            mapped = 'abnormal'
        if mapped is not None:
            mapping[fname_key] = mapped

print("Parsed labeled entries:", len(mapping))
```

```
Parsed labeled entries: 3151
```

Find audio files in extracted dataset

```
audio_exts = {'.wav', '.WAV', '.flac', '.FLAC', '.mp3', '.MP3'}
audio_files = []
for root, dirs, files in os.walk(EXTRACT_DIR):
    for f in files:
        if Path(f).suffix in audio_exts:
            audio_files.append(Path(root) / f)
audio_files = sorted(audio_files)
print("Total audio files found:", len(audio_files))
```

```
Total audio files found: 3541
```

Match mapping -> build list of labeled files (Normal/Abnormal). Skip unknown/poor.

```
label_to_int = {'normal': 0, 'abnormal': 1}
X_paths, y_labels = [], []
for p in audio_files:
    key = p.name
    if key in mapping and mapping[key] in label_to_int:
        X_paths.append(p); y_labels.append(label_to_int[mapping[key]])
# try stems if nothing matched
if len(X_paths) == 0:
    for p in audio_files:
        if p.stem in mapping and mapping[p.stem] in label_to_int:
            X_paths.append(p); y_labels.append(label_to_int[mapping[p.stem]])

if len(X_paths) == 0:
    raise RuntimeError("No labeled Normal/Abnormal audio files matched. Check annotation CSVs and filenames.")
```

```
if len(X_paths) > MAX_SAMPLES:
    X_paths = X_paths[:MAX_SAMPLES]; y_labels = y_labels[:MAX_SAMPLES]

print("Files used (Normal/Abnormal):", len(X_paths), "class_counts:", {0:int(sum(1 for v in y_labels if v==0)), 1:int(sum(1
```

```
Files used (Normal/Abnormal): 3000 class_counts: {0: 2375, 1: 625}
```

Preprocessing

```
def resample_if_needed(x, orig_sr, target_sr):
    if orig_sr == target_sr:
        return x
    target_len = int(round(len(x) * (target_sr / orig_sr)))
    return signal.resample(x, target_len).astype(np.float32)

def waveform_to_spectrogram_image(x, sr, img_h=IMG_H, img_w=IMG_W, n_fft=N_FFT, hop=HOP):
    max_len = int(MAX_SEC * TARGET_SR)
    if len(x) > max_len:
        x = x[:max_len]
    else:
        x = np.pad(x, (0, max(0, max_len - len(x))), 'constant')
    f, t, Z = signal.stft(x, fs=sr, nperseg=n_fft, noverlap=n_fft-hop)
    S = np.abs(Z)
    if S.shape[0] < img_h:
        S = np.pad(S, ((0, img_h - S.shape[0]), (0,0)), 'constant')
    else:
        S = S[:img_h, :]
    S_log = np.log1p(S)
    S_log = (S_log - S_log.min()) / (S_log.max() - S_log.min() + 1e-9)
    S_resized = signal.resample(S_log, img_w, axis=1)
    if S_resized.shape[0] != img_h:
        S_resized = signal.resample(S_resized, img_h, axis=0)
    return S_resized.astype(np.float32)
```

Build dataset (spectrogram images)

```
X_imgs = []
y_final = []
fail_count = 0
for p, lbl in zip(X_paths, y_labels):
    try:
        wav, sr = load_audio(p)
        wav = resample_if_needed(wav, sr, TARGET_SR)
        img = waveform_to_spectrogram_image(wav, TARGET_SR)
        X_imgs.append(img[..., np.newaxis])
        y_final.append(lbl)
    except Exception as e:
        fail_count += 1
        print("Failed to process", p, ":", e)
print("Built spectrogram images:", len(X_imgs), "failed:", fail_count)

X = np.stack(X_imgs).astype(np.float32)
y = np.array(y_final, dtype=np.int32)
print("X shape:", X.shape, "y shape:", y.shape)
```

```
Built spectrogram images: 3000 failed: 0
X shape: (3000, 128, 128, 1) y shape: (3000,)
```

Train/val/test split (stratified)

```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, stratify=y, random_state=RANDOM_SEED)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.50, stratify=y_temp, random_state=RANDOM_SEED)
print("Train/Val/Test shapes:", X_train.shape, X_val.shape, X_test.shape)
```

```
Train/Val/Test shapes: (2100, 128, 128, 1) (450, 128, 128, 1) (450, 128, 128, 1)
```

Models (all with Dense(2, softmax) final)

```
def build_cnn(input_shape=(IMG_H, IMG_W, 1), num_classes=2):
    inp = layers.Input(shape=input_shape)
    x = layers.Conv2D(16, 3, padding='same', activation='relu')(inp)
    x = layers.MaxPool2D()(x)
    x = layers.Conv2D(32, 3, padding='same', activation='relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Conv2D(64, 3, padding='same', activation='relu')(x)
    x = layers.GlobalAveragePooling2D()(x)
```

```
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    out = layers.Dense(2, activation='softmax')(x)   # <-- 2-output softmax
    model = models.Model(inp, out)
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

def build_transfer_mobilenet(input_shape=(IMG_H, IMG_W, 3), num_classes=2):
    base = tf.keras.applications.MobileNetV2(weights='imagenet', include_top=False, input_shape=input_shape)
    base.trainable = False
    inp = layers.Input(shape=input_shape)
    x = base(inp, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    out = layers.Dense(2, activation='softmax')(x)  # <-- 2-output softmax
    model = models.Model(inp, out)
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

def build_transfer_resnet50(input_shape=(IMG_H, IMG_W, 3), num_classes=2):
    base = tf.keras.applications.ResNet50(weights='imagenet', include_top=False, input_shape=input_shape)
    base.trainable = False
    inp = layers.Input(shape=input_shape)
    x = base(inp, training=False)
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.3)(x)
    out = layers.Dense(2, activation='softmax')(x)  # <-- 2-output softmax
    model = models.Model(inp, out)
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

TF data pipeline

```
def make_dataset(Xa, ya, batch=BATCH_SIZE, shuffle=True, for_transfer=False):
    """
    Xa: numpy array of shape (N, H, W, C). C==1 for grayscale spectrograms.
    for_transfer: if True, convert grayscale->RGB inside the pipeline. If Xa already has 3 channels,
                  set for_transfer=False to avoid double conversion.
    """
    ds = tf.data.Dataset.from_tensor_slices((Xa, ya))
    if shuffle:
        ds = ds.shuffle(1024, seed=RANDOM_SEED)
    def _map(x, y):
        x = tf.cast(x, tf.float32)
        if for_transfer:
            if tf.shape(x)[-1] == 1:
                x = tf.image.grayscale_to_rgb(x)
            # ensure proper resize (though shapes already IMG_H x IMG_W)
            x = tf.image.resize(x, [IMG_H, IMG_W])
        return x, y
    ds = ds.map(_map, num_parallel_calls=tf.data.AUTOTUNE).batch(batch).prefetch(tf.data.AUTOTUNE)
    return ds
```

Prepare datasets

```
train_ds = make_dataset(X_train, y_train, for_transfer=False)
val_ds = make_dataset(X_val, y_val, shuffle=False, for_transfer=False)
test_ds = make_dataset(X_test, y_test, shuffle=False, for_transfer=False)
```

For transfer models, either pass grayscale and let make_dataset convert (for_transfer=True)

```
train_ds_tf = make_dataset(X_train, y_train, for_transfer=True)
val_ds_tf   = make_dataset(X_val,   y_val,   shuffle=False, for_transfer=True)
test_ds_tf  = make_dataset(X_test,  y_test,  shuffle=False, for_transfer=True)
```

Robust train & evaluate

```
def train_and_evaluate(model, train_ds_local, val_ds_local, test_ds_local, model_name):
    print(f"\n=== Training {model_name} ===")
    model.fit(train_ds_local, validation_data=val_ds_local, epochs=EPOCHS)
    print(f"=== Evaluating {model_name} ===")
    y_proba = model.predict(test_ds_local)
    y_proba = np.array(y_proba)
    # normalize to 2D
    if y_proba.ndim == 1:
```

```
        y_proba = y_proba.reshape(-1, 1)
    # Determine probability of Abnormal class and predictions
    if y_proba.shape[1] == 1:
        prob_abnormal = y_proba.ravel()
        y_pred = (prob_abnormal >= 0.5).astype(int)
    else:
        prob_abnormal = y_proba[:, 1]
        y_pred = np.argmax(y_proba, axis=1)
    y_true = np.concatenate([y for x, y in test_ds_local], axis=0)

    acc = accuracy_score(y_true, y_pred)
    cm = confusion_matrix(y_true, y_pred)
    report = classification_report(y_true, y_pred, target_names=['Normal','Abnormal'], output_dict=True)

    # Save confusion matrix
    fig, ax = plt.subplots(figsize=(5,5))
    ax.imshow(cm, aspect='auto')
    ax.set_title(f"Confusion Matrix - {model_name}")
    ax.set_xlabel("Predicted"); ax.set_ylabel("True")
    ax.set_xticks([0,1]); ax.set_yticks([0,1])
    ax.set_xticklabels(['Normal','Abnormal'], rotation=45); ax.set_yticklabels(['Normal','Abnormal'])
    plt.tight_layout()
    cm_path = OUTPUT_DIR / f"confusion_{model_name}.png"
    plt.savefig(cm_path); plt.close(fig)

    # Precision-Recall for Abnormal
    ap_val = None
    try:
        y_true_bin = sklearn.preprocessing.label_binarize(y_true, classes=[0,1])
        precision, recall, _ = precision_recall_curve(y_true_bin[:,1], prob_abnormal)
        ap_val = average_precision_score(y_true_bin[:,1], prob_abnormal)
        fig, ax = plt.subplots()
        ax.plot(recall, precision)
        ax.set_xlabel("Recall"); ax.set_ylabel("Precision")
        ax.set_title(f"Precision-Recall ({model_name}) - Abnormal (AP={ap_val:.3f})")
        plt.tight_layout()
        pr_path = OUTPUT_DIR / f"pr_{model_name}_abnormal.png"
        plt.savefig(pr_path); plt.close(fig)
    except Exception as e:
        print(f"PR curve could not be computed for {model_name}: {e}")
        pr_path = None

    # Save classification report
    report_path = OUTPUT_DIR / f"classification_report_{model_name}.json"
    with open(report_path, "w") as f:
        json.dump(report, f, indent=2)

    return {
        "accuracy": float(acc),
        "ap_abnormal": (float(ap_val) if ap_val is not None else None),
        "confusion": str(cm_path.name),
        "pr": (str(pr_path.name) if pr_path is not None else None),
        "report": str(report_path.name)
    }
```

Train & evaluate models

```
summary = {"num_samples": int(len(X)), "class_counts": {"normal": int((y==0).sum()), "abnormal": int((y==1).sum())}, "models
```

```
# Robust evaluation cell (fixed for the label_binarize shape issue)
import numpy as np, json, matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_recall_curve, average_precisi
import sklearn

# Use the trained cnn model and test_ds (tf.data.Dataset)
cnn = build_cnn()
cnn.fit(test_ds, validation_data=val_ds, epochs=EPOCHS)
y_proba_raw = cnn.predict(test_ds)
y_proba = np.array(y_proba_raw)
print("DEBUG: raw y_proba.shape =", y_proba.shape)
print("DEBUG: sample y_proba (first 5 rows):\n", y_proba[:5])

# Normalize to two-column probs: [P(normal), P(abnormal)]
if y_proba.ndim == 1:
    p_pos = y_proba.ravel()
    probs = np.vstack([1.0 - p_pos, p_pos]).T
elif y_proba.ndim == 2 and y_proba.shape[1] == 1:
    p_pos = y_proba.ravel()
    probs = np.vstack([1.0 - p_pos, p_pos]).T
elif y_proba.ndim == 2 and y_proba.shape[1] >= 2:
```

```
    probs = y_proba
else:
    raise ValueError("Unexpected y_proba shape: " + str(y_proba.shape))

print("Normalized probs.shape =", probs.shape)
print("Normalized sample (first 5):\n", probs[:5])

# True labels and predicted labels
y_true = np.concatenate([y for x, y in test_ds], axis=0)
y_pred = np.argmax(probs, axis=1)

# Basic metrics
acc = accuracy_score(y_true, y_pred)
cm = confusion_matrix(y_true, y_pred)
report = classification_report(y_true, y_pred, target_names=['Normal', 'Abnormal'], output_dict=True)
print("Accuracy:", acc)
print("Confusion matrix:\n", cm)

# Save confusion matrix figure
fig, ax = plt.subplots(figsize=(5,5))
ax.imshow(cm, aspect='auto')
ax.set_title("Confusion Matrix - cnn")
ax.set_xlabel("Predicted"); ax.set_ylabel("True")
ax.set_xticks([0,1]); ax.set_yticks([0,1])
ax.set_xticklabels(['Normal','Abnormal'], rotation=45); ax.set_yticklabels(['Normal','Abnormal'])
plt.tight_layout()
plt.savefig("/mnt/data/confusion_CNN_final.png")
plt.close(fig)

# --- FIXED: build positive-class truth vector robustly ---
y_true_bin = sklearn.preprocessing.label_binarize(y_true, classes=[0,1])
# label_binarize may return shape (N,1) for binary case; handle both shapes:
if y_true_bin.ndim == 1:
    y_true_pos = y_true_bin
elif y_true_bin.ndim == 2 and y_true_bin.shape[1] == 1:
    y_true_pos = y_true_bin.ravel()
else:
    # if it returned two columns, the second column is the positive class
    y_true_pos = y_true_bin[:, 1]

# Probabilities for the Abnormal class
prob_abnormal = probs[:, 1]

# Precision-Recall and Average Precision
precision, recall, _ = precision_recall_curve(y_true_pos, prob_abnormal)
ap = average_precision_score(y_true_pos, prob_abnormal)
fig, ax = plt.subplots()
ax.plot(recall, precision)
ax.set_xlabel("Recall"); ax.set_ylabel("Precision")
ax.set_title(f"Precision-Recall (cnn) - Abnormal (AP={ap:.3f})")
plt.tight_layout()
plt.savefig("/mnt/data/pr_CNN_final.png")
plt.close(fig)

# Save classification report
with open("/mnt/data/classification_report_CNN_final.json", "w") as f:
    json.dump(report, f, indent=2)

print("Saved: /mnt/data/confusion_CNN_final.png, /mnt/data/pr_CNN_final.png, /mnt/data/classification_report_CNN_final.json"
print("AP (Abnormal) =", ap)
```

```
Epoch 1/6
29/29 ━━━━━━━━━━━━━━━━━━━━ 18s 495ms/step - accuracy: 0.7227 - loss: 0.6361 - val_accuracy: 0.7911 - val_loss: 0.5027
Epoch 2/6
29/29 ━━━━━━━━━━━━━━━━━━━━ 8s 296ms/step - accuracy: 0.7810 - loss: 0.5289 - val_accuracy: 0.7911 - val_loss: 0.5002
Epoch 3/6
29/29 ━━━━━━━━━━━━━━━━━━━━ 8s 262ms/step - accuracy: 0.7810 - loss: 0.5317 - val_accuracy: 0.7911 - val_loss: 0.4956
Epoch 4/6
29/29 ━━━━━━━━━━━━━━━━━━━━ 8s 294ms/step - accuracy: 0.7810 - loss: 0.5320 - val_accuracy: 0.7911 - val_loss: 0.4908
Epoch 5/6
29/29 ━━━━━━━━━━━━━━━━━━━━ 8s 271ms/step - accuracy: 0.7810 - loss: 0.5270 - val_accuracy: 0.7911 - val_loss: 0.4865
Epoch 6/6
29/29 ━━━━━━━━━━━━━━━━━━━━ 9s 235ms/step - accuracy: 0.7810 - loss: 0.5233 - val_accuracy: 0.7911 - val_loss: 0.4811
29/29 ━━━━━━━━━━━━━━━━━━━━ 2s 57ms/step
DEBUG: raw y_proba.shape = (450, 2)
DEBUG: sample y_proba (first 5 rows):
 [[0.80200416 0.19799583]
 [0.7852961  0.21470392]
 [0.7816575  0.21834233]
 [0.9389903  0.06100964]
 [0.8643422  0.1356578 ]]
Normalized probs.shape = (450, 2)
Normalized sample (first 5):
 [[0.80200416 0.19799583]
 [0.7852961  0.21470392]
 [0.7816575  0.21834233]
 [0.9389903  0.06100964]
 [0.8643422  0.1356578 ]]
Accuracy: 0.7933333333333333
Confusion matrix:
 [[357   0]
 [ 93   0]]
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
--------------------------------------------------------------------------------
```