**TRƯỜNG ĐẠI HỌC THỦY LỢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**----- & -----**

****

| **MÔN: KHAI PHÁ DỮ LIỆU**    **ĐỀ TÀI**  **PHÂN LỚP: PHÂN LOẠI ÂM THANH**  **Nhóm thực hiện: 64HTTT3-9**  **Thành viên nhóm:** *1.* ***Bùi Tuấn Kiên*** *– MSV: 2251162046  2.* ***Trần Thu Hương*** *– MSV: 2251162031 3.* ***Đào Công Minh Vũ*** *– MSV: 2251162211*  *4.* ***Đỗ Văn An*** *– MSV: 2251161932*  *5.* ***Hồ Bá Vũ Tùng*** *– MSV: 2251162206*  *6.*  ***Đào Hải Phúc*** *– MSV: 2151163713* |
| --- |
|  |

Hà Nội, 2025

**Bảng phân chia công việc**

| **Công việc** | **Người phụ trách** |
| --- | --- |
| Làm báo cáo | Cả nhóm |
| Tiền xử lý dữ liệu | Đỗ Văn An, Hồ Bá Vũ Tùng |
| Xây dựng thuật toán KNN | Trần Thu Hương, Bùi Tuấn Kiên |
| Xây dựng thuật toán RF | Đào Công Minh Vũ, Đào Hải Phúc |
| Xây dựng thuật toán CNN | Cả nhóm |

**Mục Lục**

[**Lời mở đầu 5**](#_jxaiazmlo4kj)

[**Chương I. Mô tả bài toán 6**](#_77fxrj4pyj7t)

[1. Tổng quan bài toán 6](#_aah9e8uiyjku)

[2. Phân tích dữ liệu 6](#_4tbcshxt5rl9)

[3. Thông tin thuộc tính 8](#_e54cfon8a7p)

[4. Trực quan hóa trước khi xử lý dữ liệu 8](#_st80ukijyux5)

[**Chương II. Tiền xử lý dữ liệu 11**](#_oum6yik4nc58)

[1. Import thư viện 11](#_dyn1elvoqw55)

[2. Kích thước dữ liệu 11](#_mioz3qmvsrqb)

[3. Kiểm tra kiểu dữ liệu các cột 11](#_gfpeean0z4gx)

[4. Kiểm tra dữ liệu thiếu 12](#_dlxuhqqogmqy)

[5. Kiểm tra file âm thanh 13](#_nrgaxcvbxtj4)

[6. Xây dựng các hàm trích xuất đặc trưng 14](#_13zhdpgusnr)

[7. Hiển thị dữ liệu và lưu file dữ liệu sau khi được trích xuất 15](#_c2zf77elt7gg)

[8. Mã hóa nhãn 16](#_w8e0v1dwh22i)

[9. Chuẩn hóa đặc trưng 17](#_u6flntk93ye2)

[**Chương III. Trực quan hóa dữ liệu 19**](#_akqhybe1e2np)

[Bảng trực quan hóa : 19](#_k1s693iztrsw)

[**Chương IV. Xây dựng thuật toán KNN 23**](#_k3254gsb1plq)

[1. Lý thuyết 23](#_qctpk52bmvki)

[2. Các bước của thuật toán 23](#_sj6wpmorbu8z)

[3. Xây dựng thuật toán 24](#_iw5374bngihz)

[4. Trực quan hóa kết quả dự đoán và đánh giá mô hình 31](#_m4ogg066e275)

[**Chương V. Xây dựng thuật toán Random Forest (RF) 33**](#_lxu9965btpbv)

[1. Lý thuyết 33](#_38c4supgh9d)

[Ngẫu nhiên trong chọn mẫu 33](#_2hlbwq6myhp1)

[Ngẫu nhiên trong chọn đặc trưng 33](#_hnx6sxn7bvi7)

[2. Các bước của thuật toán 34](#_ugqof7phbubw)

[3. Xây dựng thuật toán 34](#_7moakoj0svk6)

[4. Trực quan hóa và đánh giá chi tiết hiệu suất mô hình Random Forest bằng các biểu đồ và số liệu 37](#_tqzjrtgaczs)

[**Chương VI. Xây dựng thuật toán CNN 40**](#_3duicv7zximf)

[1. Lý thuyết 40](#_y5lhleixczvu)

[2. Các bước của thuật toán 40](#_pbspbhk2jsqe)

[3. Xây dựng thuật toán 42](#_qmgkmffn3v)

[**Chương VII. Kết luận và hướng phát triển 53**](#_l86ktrdkgv3o)

[1) Phân tích và đánh giá kết quả 53](#_fc37428m7wgz)

[1.1. KNN (K-Nearest Neighbors) 53](#_ql1z56t02ikd)

[1.2. Random Forest 53](#_egsrbhg7n4jk)

[1.3. CNN (Convolutional Neural Network) 53](#_uppq5p4j3wxy)

[2) Đánh giá mô hình 54](#_vq95cx57vzzi)

[3) Hướng phát triển 55](#_dr53un10n6dl)

# 

# 

# 

# 

# **Lời mở đầu**

Trong thời đại công nghệ số hiện nay, âm thanh là một trong những nguồn dữ liệu quan trọng và phổ biến bên cạnh hình ảnh và văn bản. Âm thanh xuất hiện khắp mọi nơi trong cuộc sống hàng ngày – từ âm thanh giao thông, tiếng người nói, tiếng động vật, đến âm thanh trong môi trường tự nhiên hoặc công nghiệp. Việc phân loại âm thanh tự động ngày càng trở nên cần thiết, phục vụ cho nhiều mục đích như: giám sát an ninh, chăm sóc sức khỏe, nhà thông minh, giao tiếp người – máy, và hỗ trợ người khuyết tật.

Tuy nhiên, việc xử lý và phân loại âm thanh là một bài toán thách thức do tín hiệu âm thanh có đặc điểm phi tuyến, nhiễu cao, đa dạng về nguồn và ngữ cảnh. Điều này đòi hỏi sự kết hợp giữa các kỹ thuật xử lý tín hiệu số và trí tuệ nhân tạo, đặc biệt là học máy và học sâu.

Vì vậy, nhóm em chọn đề tài **“Phân loại âm thanh bằng phương pháp học máy”** với mong muốn tìm hiểu quy trình xử lý tín hiệu âm thanh, trích xuất đặc trưng, huấn luyện mô hình phân loại, từ đó ứng dụng vào các tình huống thực tiễn như nhận diện môi trường, cảnh báo âm thanh bất thường hay xây dựng hệ thống hỗ trợ thông minh.

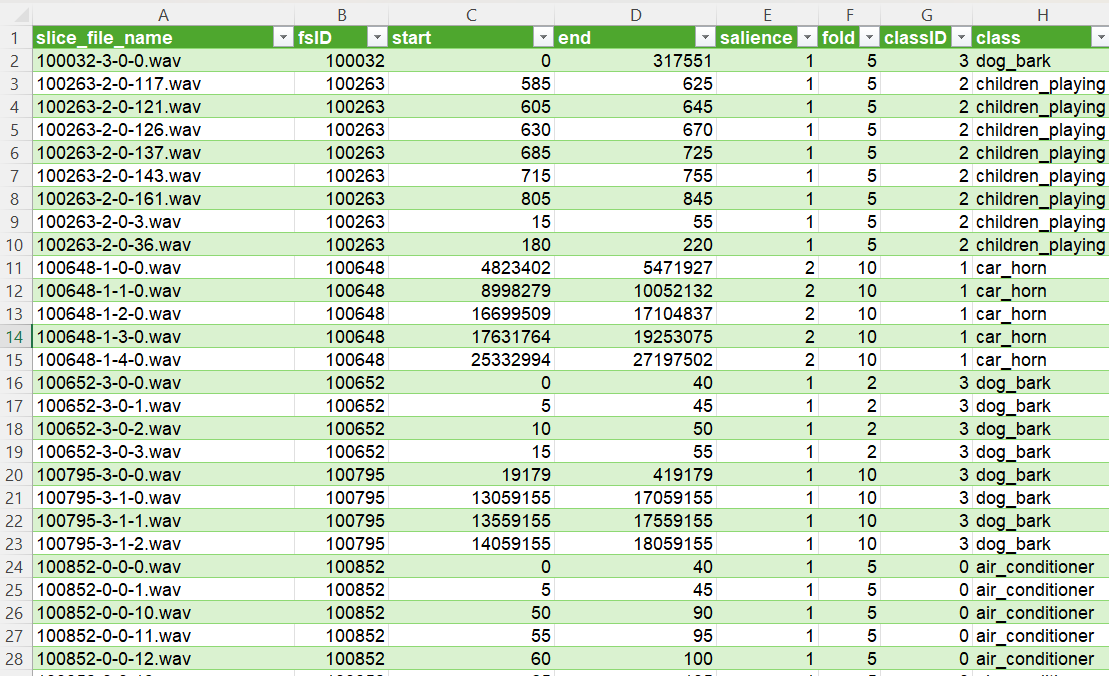
# **Chương I. Mô tả bài toán**

## **Tổng quan bài toán**

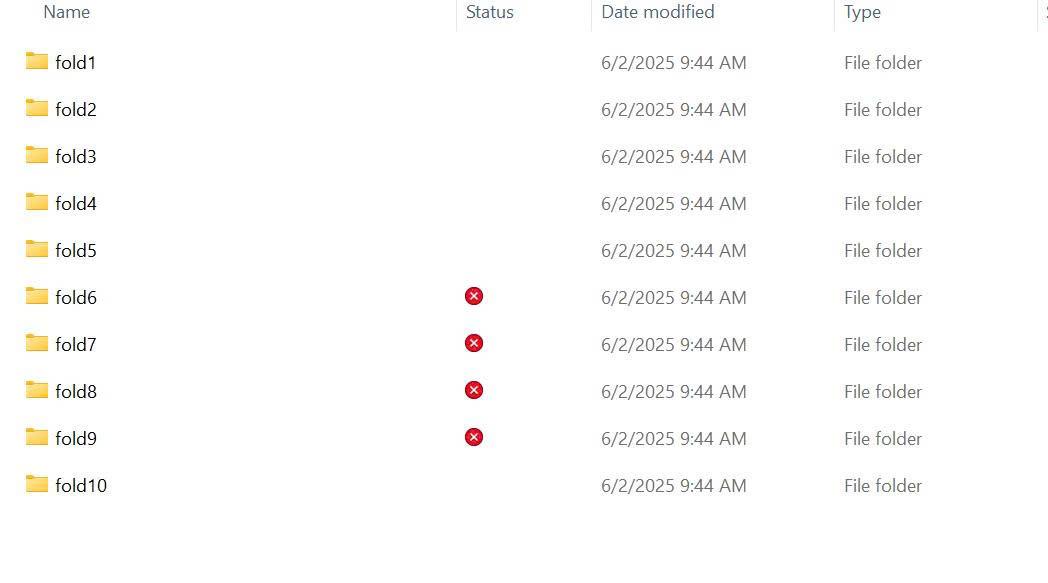
Bài toán phân loại âm thanh sử dụng tập dữ liệu UrbanSound8K là một bài toán học máy nhằm xác định loại âm thanh xuất hiện trong một đoạn ghi âm ngắn. UrbanSound8K gồm 8.732 tệp âm thanh thuộc 10 lớp âm thanh đô thị phổ biến như tiếng chó sủa, tiếng còi xe, tiếng khoan, tiếng nhạc đường phố,... Mỗi đoạn âm thanh có độ dài không quá 4 giây và được gắn nhãn sẵn, hỗ trợ cho việc huấn luyện và đánh giá mô hình. Mục tiêu của bài toán là xây dựng một mô hình có thể nhận diện chính xác loại âm thanh đầu vào bằng cách trích xuất các đặc trưng như MFCC, ZCR, RMS rồi đưa vào các thuật toán học máy như KNN, Logistic Regression hoặc mạng nơ-ron tích chập (CNN). Bài toán này có tính ứng dụng cao trong nhiều lĩnh vực như giám sát đô thị, cảnh báo an ninh, hay phân tích môi trường âm thanh thông minh.

## **Phân tích dữ liệu**

* **Nguồn dữ liệu:** [UrbanSound8K](https://www.kaggle.com/datasets/chrisfilo/urbansound8k/code?fbclid=IwY2xjawK6FaxleHRuA2FlbQIxMABicmlkETFqOUV0MWl5blREMUhTU3JsAR60D-XdopYOWefEdXFzjeyiqnhlhk7_2XtAUvNoNPdtd3W-sl_uqOhOD_ZCBA_aem_00JkUcvTmLcJ7sTGMpfzvA) - tập dữ liệu âm thanh dạng thô có nhãn.
* **Hiểu dữ liệu** **:** Dữ liệu hỗ trợ nghiên cứu nhận dạng âm thanh trong môi trường đô thị. Các đoạn âm thanh được thu thập từ kho Freesound.org, mỗi đoạn có độ dài không quá 4 giây nhằm tập trung vào phân loại các âm thanh ngắn mang tính đặc trưng của môi trường đô thị.
* **Thành phần dữ liệu gồm :**
* Bao gồm **8.732** tệp âm thanh định dạng  **.wav**, được gán nhãn tương ứng với **10 lớp âm thanh** khác nhau: điều hòa, còi xe, trẻ chơi, chó sủa, khoan, máy chạy không tải, súng nổ, máy đục, còi cảnh sát / xe cấp cứu, nhạc đường phố
* Một file csv UrbanSound8K – chứa thông tin mô tả chi tiết về từng đoạn âm thanh trong tập dữ liệu..



*Hình 1 : Một phần dữ liệu metadata từ tập UrbanSound8K (file .csv), chứa tên file, nhãn âm thanh, vị trí fold và các thông tin bổ sung khác.*

**

*Hình 2 : Cấu trúc thư mực UrbanSound8K, mỗi fold chứa một phần dữ liệu âm thanh phục vụ huấn luyện kiểm thử*

## **Thông tin thuộc tính**

| **STT** | **Thuộc tính** | **Ý nghĩa** |
| --- | --- | --- |
| 1 | slice\_file\_name | Tên file âm thanh chứa đoạn trích tương ứng. |
| 2 | fsID | Mã ID của nguồn âm thanh từ Freesound.org |
| 3 | start | Thời gian bắt đầu của đoạn âm thanh |
| 4 | end | Thời gian kết thúc của đoạn âm thanh |
| 5 | salience | Mức độ nổi bật (salience) của sự kiện âm thanh, biểu thị mức độ quan trọng hoặc rõ ràng của âm thanh |
| 6 | fold | Số thứ tự của thư mục mà đoạn âm thanh thuộc về. |
| 7 | classID | Mã số định danh của lớp âm thanh |
| 8 | class | Tên lớp âm thanh |

# 

# 

# 

# 

# 

# 

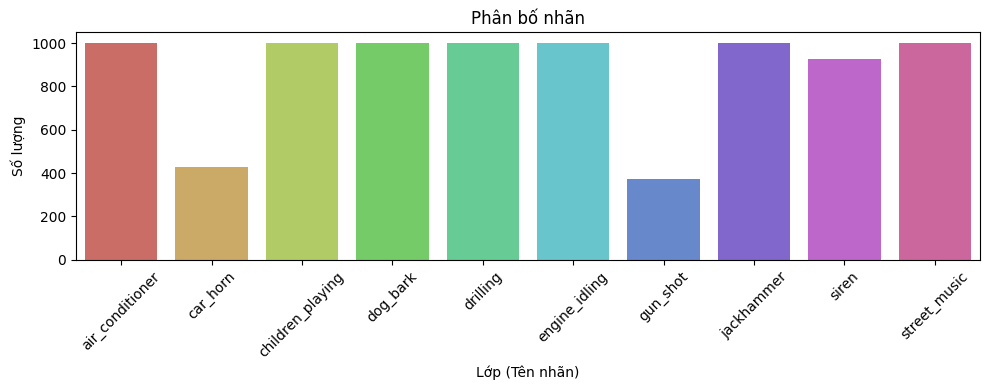
# 

# 

# 

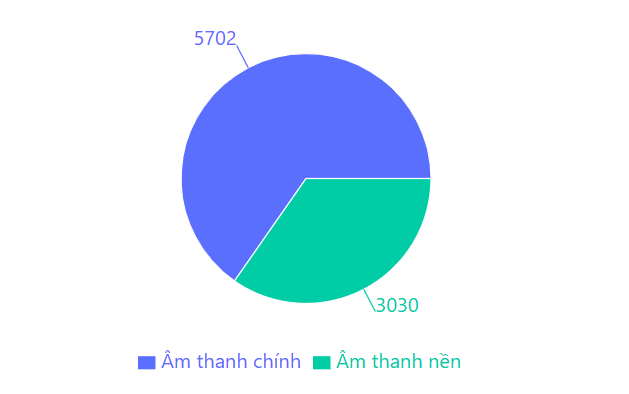
## **Trực quan hóa trước khi xử lý dữ liệu**

1. **Phân bố âm thanh trong dữ liệu**



*Hình 3:* Phân bố âm thanh trong dữ liệu

1. **Phân bố độ nổi bật**



*Hình 4:* Phân bố độ nổi bật

* Tỷ lệ giữa âm thanh chính và nền có thể gần cân bằng, cho thấy bộ dữ liệu bao gồm cả các bản ghi rõ ràng và các bản ghi có nhiễu nền.

1. **Phân bố thời lượng trung bình theo âm thanh**



*Hình 5:* Phân bố thời lượng trung bình theo âm thanh

* Biểu đồ cho thấy thời lượng âm thanh phần lớn dao động quanh 4 giây, nhiều loại âm có độ lệch lớn như **car\_horn**, **dog\_bark** và **gun\_shot**. Một số âm như **air\_conditioner**, **street\_music** có thời lượng ổn định hơn, gần sát trần 4 giây. Các điểm ngoại lai xuất hiện khá phổ biến ở hầu hết các loại âm.

1. **Phân bố nhãn theo Fold**

### 

*Hình 6:* Phân bố nhãn theo fold

* Biểu đồ thể hiện số lượng mẫu phân bố tương đối đồng đều giữa các Fold. Fold 4 có số mẫu cao nhất (~990), trong khi Fold 8 thấp nhất (~810), chênh lệch không quá lớn, cho thấy dữ liệu đã được chia khá cân bằng.

# **Chương II. Tiền xử lý dữ liệu**

## **Import thư viện**

| import os  import numpy as np  import pandas as pd  import librosa  import librosa.display  import matplotlib.pyplot as plt  import seaborn as sns  import IPython.display as ipd  from collections import Counter  from sklearn.metrics import confusion\_matrix  from sklearn.model\_selection import train\_test\_split, StratifiedKFold  from sklearn.metrics import classification\_report, accuracy\_score  import warnings  warnings.filterwarnings("ignore", category=UserWarning) |
| --- |

## **Kích thước dữ liệu**

# 

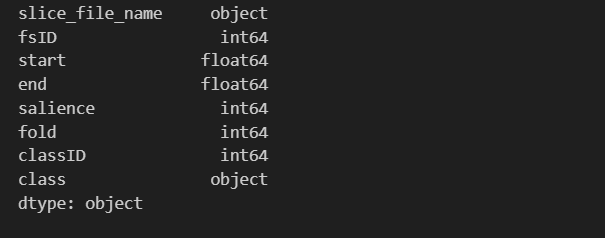


*Hình 8 + 9:* Kích thước dữ liệu

Dữ liệu có 8732 dòng và 8 cột

## **Kiểm tra kiểu dữ liệu các cột**

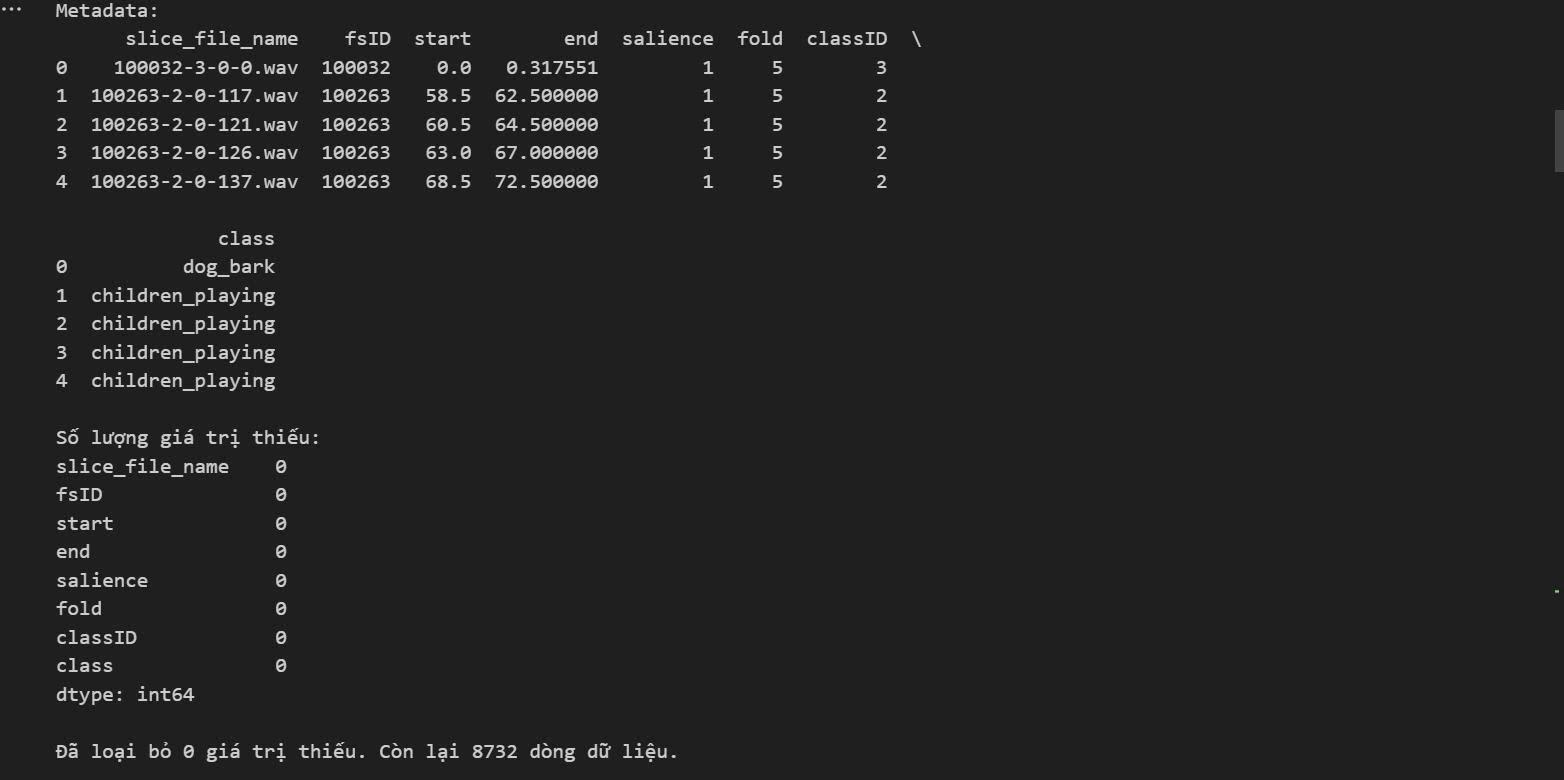




*Hình 10 + 11:* Kiểm tra dữ liệu các cột

## **Kiểm tra dữ liệu thiếu**

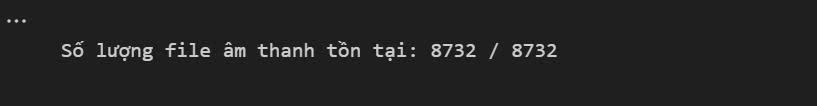
| # Cấu hình đường dẫn  BASE\_DIR = r'C:\Users\ADMIN\OneDrive\Documents\BTAPKPDL\cuoiki\code\archive'  CSV\_PATH = os.path.join(BASE\_DIR, 'metadata', 'UrbanSound8K.csv')  AUDIO\_BASE\_DIR = os.path.join(BASE\_DIR, 'audio')  # Đọc metadata  df = pd.read\_csv(CSV\_PATH)  print("Metadata:")  print(df.head())  # Kiểm tra dữ liệu thiếu  missing\_values = df.isnull().sum()  print("\nSố lượng giá trị thiếu:")  print(missing\_values)  df = df.dropna().reset\_index(drop=True)  print(f"\nĐã loại bỏ {missing\_values.sum()} giá trị thiếu. Còn lại {len(df)} dòng dữ liệu.") |
| --- |



*Hình 12:* Kiểm tra dữ liệu thiếu

## **Kiểm tra file âm thanh**

| # Kiểm tra file âm thanh có tồn tại không  def check\_existing\_files(df, audio\_base\_dir):  def file\_exists(row):  fold = f"fold{row['fold']}"  filename = row['slice\_file\_name']  file\_path = os.path.join(audio\_base\_dir, fold, filename)  return os.path.isfile(file\_path)    df['file\_exists'] = df.apply(file\_exists, axis=1)  return df[df['file\_exists']].reset\_index(drop=True)  df\_exist = check\_existing\_files(df, AUDIO\_BASE\_DIR)  print(f"\nSố lượng file âm thanh tồn tại: {len(df\_exist)} / {len(df)}") |
| --- |



*Hình 13:* Kiểm tra file âm thanh

## **Xây dựng các hàm trích xuất đặc trưng**

Trong bài toán phân loại âm thanh, file .wav là tín hiệu dạng sóng thô, chứa hàng nghìn giá trị biên độ theo thời gian – không thể đưa trực tiếp vào mô hình học máy. Vì vậy, cần trích xuất các đặc trưng âm thanh nhằm rút gọn và mô tả bản chất của âm thanh một cách hiệu quả:

MFCC (Mel-Frequency Cepstral Coefficients): Mô tả phổ tần số theo cách tai người cảm nhận, thường dùng trong nhận diện giọng nói và tiếng động môi trường.

ZCR (Zero-Crossing Rate): Biểu diễn tần suất tín hiệu đổi dấu, phản ánh âm thanh có tần số cao hay thấp.

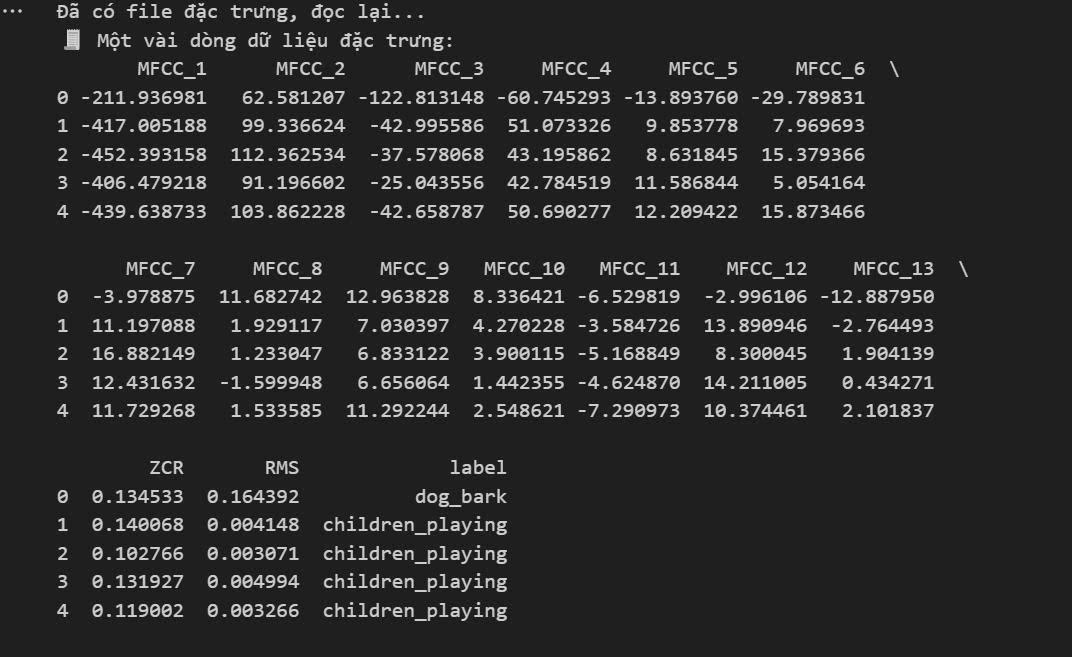
RMS (Root Mean Square Energy): Đo năng lượng trung bình của âm thanh, giúp phân biệt tiếng to nhỏ, foreground/background.

Việc trích xuất các đặc trưng này giúp chuyển đổi tín hiệu âm thanh thành dạng vector số ngắn gọn, phù hợp để huấn luyện các mô hình học máy.

| # Xây dựng các hàm trích xuất đặc trưng  def extract\_mfcc(y, sr, n\_mfcc=13):  mfcc = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=n\_mfcc)  return np.mean(mfcc, axis=1)  def extract\_zcr(y):  zero\_crossings = np.where(np.diff(np.sign(y)))[0]  return len(zero\_crossings) / len(y)  def extract\_rms(y):  return np.sqrt(np.mean(np.square(y)))  def extract\_audio\_features(file\_path):  y, sr = load\_audio(file\_path)  if y is None:  return None  try:  mfcc = extract\_mfcc(y, sr)  zcr = extract\_zcr(y)  rms = extract\_rms(y)  return np.hstack([mfcc, zcr, rms])  except Exception as e:  print(f"Lỗi đặc trưng: {file\_path} | {e}")  return None |
| --- |

## **Hiển thị dữ liệu và lưu file dữ liệu sau khi được trích xuất**

| FEATURES\_PATH = os.path.join(BASE\_DIR, 'metadata', "extracted\_features.csv")  if os.path.exists(FEATURES\_PATH):  print("Đã có file đặc trưng, đọc lại...")  df\_features = pd.read\_csv(FEATURES\_PATH)  features = df\_features.drop(columns=['label']).values  labels = df\_features['label'].values  else:  print("Chưa có file đặc trưng, đang trích xuất...")    def process\_features(df\_exist):  features, labels = [], []  for idx, row in df\_exist.iterrows():  fold = f"fold{row['fold']}"  filename = row['slice\_file\_name']  file\_path = os.path.join(AUDIO\_BASE\_DIR, fold, filename)  feat = extract\_audio\_features(file\_path)  if feat is not None:  features.append(feat)  labels.append(row['class'])  return np.array(features), np.array(labels)  features, labels = process\_features(df\_exist)  df\_features = pd.DataFrame(  features,  columns=[f'MFCC\_{i+1}' for i in range(13)] + ['ZCR', 'RMS']  )  df\_features['label'] = labels  df\_features.to\_csv(FEATURES\_PATH, index=False, encoding='utf-8-sig')  print(f"Đã lưu đặc trưng vào: {FEATURES\_PATH}")  print("🧾 Một vài dòng dữ liệu đặc trưng:")  print(df\_features.head()) |
| --- |



*Hình 14:* Dữ liệu sau khi được trích xuất đặc trưng

## **Mã hóa nhãn**

Dù file CSV đã có nhãn (class), ta vẫn dùng hàm encode\_labels() để mã hóa lại nhãn bằng Label Encoding nhằm đảm bảo nhãn liên tục, tương thích với mô hình học máy, dễ kiểm soát thứ tự và giải mã kết quả.

| # Mã hóa nhãn  # Mặc dù trong file csv có nhãn rồi nma muốn kiểm soát thứ tự nhãn hoặc giải thích kết quả rõ hơn  def encode\_labels(labels):  classes = sorted(set(labels))  label\_map = {label: idx for idx, label in enumerate(classes)}  encoded = np.array([label\_map[label] for label in labels])  return encoded, label\_map  labels\_encoded, label\_map = encode\_labels(labels)  print("Mã hóa nhãn:")  print(pd.DataFrame({'label': labels\_encoded}).head(10))  print("Các nhãn duy nhất:", np.unique(labels\_encoded)) |
| --- |

# 

*Hình 15 : Các nhãn sau khi được mã hóa*

## **Chuẩn hóa đặc trưng**

Sử dụng hàm standardize\_features() với công thức (X - mean) / std để chuẩn hóa tất cả đặc trưng về cùng thang đo (trung bình 0, độ lệch chuẩn 1). Điều này giúp mô hình học ổn định hơn và cải thiện hiệu quả huấn luyện.

| # Chuẩn hóa đặc trưng  def standardize\_features(X):  mean = np.mean(X, axis=0)  std = np.std(X, axis=0)  std[std == 0] = 1  return (X - mean) / std, mean, std  features\_scaled, mean, std = standardize\_features(features)  # Tạo DataFrame với dữ liệu đã chuẩn hóa  df\_scaled = pd.DataFrame(features\_scaled, columns=[f'MFCC\_{i+1}' for i in range(13)] + ['ZCR', 'RMS'])  df\_scaled['label'] = labels\_encoded  output\_scaled\_path = os.path.join(BASE\_DIR, 'metadata', "standardized\_features.csv")  df\_scaled.to\_csv(output\_scaled\_path, index=False, encoding='utf-8-sig')  print(f"Đã lưu đặc trưng chuẩn hóa vào file: {output\_scaled\_path}")  print("Sau chuẩn hóa:")  print(pd.DataFrame(features\_scaled).head()) |
| --- |

### 

*Hình 16 : Các đặc trưng sau khi được chuẩn hóa*

# 

# **Chương III. Trực quan hóa dữ liệu**

## **Bảng trực quan hóa** **:**

|  | Biểu đồ MFCC\_1 sau chuẩn hóa cho thấy mỗi lớp âm thanh có phân bố khác nhau. Một số lớp dễ phân biệt, nhưng một số như **jackhammer, children\_playing** có nhiều nhiễu. Chuẩn hóa giúp dữ liệu đồng nhất và hỗ trợ mô hình học hiệu quả hơn. |
| --- | --- |
|  | Biểu đồ MFCC\_2 sau chuẩn hóa tiếp tục cho thấy sự khác biệt phân bố giữa các lớp âm thanh. Một số lớp như **siren** và **street\_music** có giá trị MFCC\_2 cao, trong khi **drilling, jackhammer** có giá trị thấp hơn. Nhiều lớp vẫn tồn tại ngoại lệ, đặc biệt là **air\_conditioner** và **children\_playing**. Chuẩn hóa giúp làm rõ sự khác biệt này, hỗ trợ mô hình học tốt hơn. |
|  | Biểu đồ cho thấy đặc trưng **MFCC\_3** (sau chuẩn hóa) có trung vị gần 0 ở hầu hết các lớp, nhưng mức độ phân tán và số lượng outliers khác nhau. Một số lớp như **air\_conditioner** và **engine\_idling** nghiêng về phía giá trị dương, trong khi **siren** và **children\_playing** có phân bố rộng, cho thấy đặc trưng này có thể giúp phân biệt giữa các lớp âm thanh. |
|  | Biểu đồ cho thấy đặc trưng **MFCC\_4** sau chuẩn hóa có trung vị gần 0 ở hầu hết các lớp, nhưng mức độ phân tán khá khác nhau. Các lớp như **dog\_bark**, **siren** và **engine\_idling** có độ biến động lớn và nhiều outliers, cho thấy đặc trưng này cũng mang thông tin phân biệt giữa các lớp âm thanh. |
|  | Biểu đồ MFCC\_5 cho thấy giá trị trung vị của các lớp vẫn xoay quanh 0, nhưng mức độ phân tán rõ rệt ở một số lớp như **engine\_idling**, **drilling**, và **siren**. Các lớp này cũng có nhiều outliers, cho thấy **MFCC\_5** góp phần phân biệt đặc điểm âm thanh giữa các lớp. |
|  | Biểu đồ MFCC\_6 cho thấy phần lớn các lớp có trung vị gần 0, nhưng mức phân tán và số lượng outliers khá khác nhau. Lớp **siren** và **street\_music** có nhiều outliers âm, trong khi **drilling** có outlier dương rõ rệt, cho thấy đặc trưng MFCC\_6 cũng giúp phân biệt giữa các lớp âm thanh. |
|  | Biểu đồ MFCC\_7 cho thấy hầu hết các lớp có trung vị gần 0, nhưng lớp **siren** nổi bật với giá trị trung vị cao và nhiều outliers dương. Điều này cho thấy đặc trưng MFCC\_7 có tiềm năng phân biệt rõ lớp **siren** so với các lớp âm thanh còn lại. |
|  | Biểu đồ MFCC\_8 tiếp tục cho thấy sự khác biệt giữa các lớp. Đặc biệt, lớp **siren** có giá trị trung vị cao và phân bố nghiêng về phía dương, kèm nhiều outliers. Điều này cho thấy **MFCC\_8** là đặc trưng giàu thông tin để phân biệt lớp **siren** với các lớp âm thanh khác. |
|  | Biểu đồ MFCC\_9 cho thấy sự khác biệt rõ rệt giữa các lớp. Các lớp như **engine\_idling**, **gun\_shot**, và **street\_music** có giá trị trung vị cao hơn và phân bố lệch về phía dương, trong khi lớp **car\_horn** có trung vị thấp hơn. Điều này cho thấy **MFCC\_9** là đặc trưng hữu ích giúp phân biệt một số lớp. |
|  | Biểu đồ MFCC\_10 cho thấy phần lớn các lớp có trung vị dương và phân bố khá đồng đều. Tuy nhiên, lớp **dog\_bark** xuất hiện nhiều outliers âm mạnh, trong khi **siren** và **street\_music** có xu hướng giá trị cao hơn. Điều này cho thấy **MFCC\_10** giúp nhận diện một số lớp có đặc trưng nổi bật. |
|  | Biểu đồ MFCC\_11 cho thấy sự phân bố khá tương đồng giữa các lớp, nhưng lớp **siren** có trung vị cao nổi bật, trong khi **car\_horn** và **jackhammer** có xu hướng thấp hơn. Điều này cho thấy **MFCC\_11** vẫn mang giá trị phân biệt nhất định, đặc biệt với lớp **siren**. |
|  | Biểu đồ MFCC\_12 cho thấy sự khác biệt rõ giữa các lớp, đặc biệt là **siren** và **street\_music** có giá trị trung bình cao hơn, trong khi **car\_horn** và **gun\_shot** có xu hướng thấp hơn. Điều này cho thấy MFCC\_12 có tiềm năng tốt trong việc phân biệt một số loại âm thanh. |
|  | MFCC\_13 phân biệt tốt các lớp như **siren**, **street\_music**, và **gun\_shot**; trong đó **siren** và **street\_music** có giá trị cao, còn **gun\_shot** phân bố hẹp. |
|  | ZCR phân biệt tốt **drilling** và **jackhammer** với giá trị cao và phân bố rộng; các lớp như **engine\_idling**, **air\_conditioner** có ZCR thấp, ổn định. |
|  | **RMS** giúp phân biệt tốt các lớp âm thanh to và ngắt quãng như **gun\_shot**, **dog\_bark**, **engine\_idling** với giá trị lớn; trong khi các lớp như **children\_playing**, **air\_conditioner** có RMS thấp hơn, ổn định. |

# 

# 

# 

# 

# 

# **Chương IV. Xây dựng thuật toán KNN**

## **Lý thuyết**

* KNN (K-Nearest Neighbors) là thuật toán phân loại dựa trên khoảng cách. Một mẫu mới sẽ được gán nhãn dựa vào đa số nhãn của K mẫu gần nhất trong không gian đặc trưng. Khoảng cách phổ biến dùng là Euclidean. Đây là thuật toán không huấn luyện (lazy learning) – toàn bộ quá trình "học" diễn ra khi dự đoán.
* Mục đích: Áp dụng KNN để phân loại âm thanh thành 1 trong 10 lớp như "dog\_bark", "siren", "drilling",… từ các đặc trưng đã trích xuất (MFCC, ZCR, RMS). Mục tiêu là xác định âm thanh đầu vào thuộc loại nào.
* Input: Ma trận đặc trưng chuẩn hóa từ các file âm thanh (.wav).
* Nhãn tương ứng với mỗi mẫu (đã mã hóa).
* Một mẫu âm thanh mới cần dự đoán.
* Giá trị K (số hàng xóm cần xét).
* Output: Nhãn âm thanh dự đoán (dạng số hoặc dạng chuỗi: "car\_horn", "engine\_idling",...).

## **Các bước của thuật toán**

* Bước 1: Tính khoảng cách

Phổ biến nhất là khoảng cách Euclidean:

|  |
| --- |

Ngoài ra có thể dùng Manhattan, Cosine,… tùy bài toán.

* Bước 2: Lựa chọn tham số k
* k: số lượng hàng xóm gần nhất để xét khi phân loại.
* k quá nhỏ → dễ bị nhiễu (overfitting).
* k quá lớn → mất chi tiết phân biệt (underfitting).

Tìm giá trị k tốt nhất bằng Cross-validation.

* Bước 3: Dự đoán nhãn cho mẫu mới
* Với mỗi điểm cần dự đoán:
* Tính khoảng cách từ điểm đó đến tất cả điểm trong tập huấn luyện.
* Chọn k điểm gần nhất.
* Dự đoán nhãn là nhãn phổ biến nhất trong k hàng xóm.
* Bước 4: Đánh giá mô hình
* So sánh nhãn dự đoán với nhãn thật trên tập kiểm tra.
* Dùng các chỉ số:

Accuracy

Precision / Recall / F1-score

Confusion Matrix

* Bước 5: Dự đoán trên dữ liệu mới

Dùng mô hình đã huấn luyện để phân loại dữ liệu mới (chưa biết nhãn).

## **Xây dựng thuật toán**

**Bước 1: Xây dựng hàm tính khoảng cách**

* Ý tưởng:

Tính khoảng cách giữa tất cả các điểm trong tập test với tất cả các điểm trong tập train một cách đồng thời, mà không cần vòng lặp for lồng nhau.

* Lý do sử dụng:

Giúp tối ưu hiệu suất: Tính toán bằng vector và ma trận nhanh hơn nhiều so với duyệt từng cặp điểm.

* Dễ triển khai: Áp dụng được cho cả tập dữ liệu lớn trong vài dòng code.
* Tận dụng được khả năng tính toán song song của thư viện như NumPy.

| # Hàm khoảng cách Euclidean dùng ma trận  def euclidean\_distance\_matrix(X1, X2):  X1 = np.array(X1)  X2 = np.array(X2)  X1\_sq = np.sum(X1\*\*2, axis=1).reshape(-1, 1)  X2\_sq = np.sum(X2\*\*2, axis=1).reshape(1, -1)  cross = np.dot(X1, X2.T)  return np.sqrt(X1\_sq + X2\_sq - 2 \* cross) |
| --- |

**Bước 2: Xây dựng hàm tìm k tốt nhất bằng Stratified K-Fold Cross Validation**

* Ý tưởng:  
   Thử nhiều giá trị k khác nhau (ví dụ từ 1 đến 10), đánh giá mô hình KNN với từng k bằng cách chia dữ liệu thành nhiều phần nhỏ (folds) và đo độ chính xác trung bình.
* Mục đích:  
   Giúp chọn được giá trị k tối ưu – tức là k cho kết quả phân loại tốt nhất, tránh quá khớp (overfitting) khi k quá nhỏ hoặc quá đơn giản (underfitting) khi k quá lớn.

| # Tìm k tốt nhất  def find\_best\_k(X, y, k\_values, n\_splits=5):  skf = StratifiedKFold(n\_splits=n\_splits, shuffle=True, random\_state=42)  best\_k, best\_acc = None, 0  for k in k\_values:  acc\_scores = []  for train\_idx, val\_idx in skf.split(X, y):  model = CustomKNN(k)  model.fit(X[train\_idx], y[train\_idx])  pred = model.predict(X[val\_idx])  acc\_scores.append(accuracy\_score(y[val\_idx], pred))  avg\_acc = np.mean(acc\_scores)  print(f"K={k}, Accuracy={avg\_acc:.4f}")  if avg\_acc > best\_acc:  best\_k, best\_acc = k, avg\_acc  return best\_k |
| --- |

**Bước 3: Xây dựng mô hình KNN thủ công**

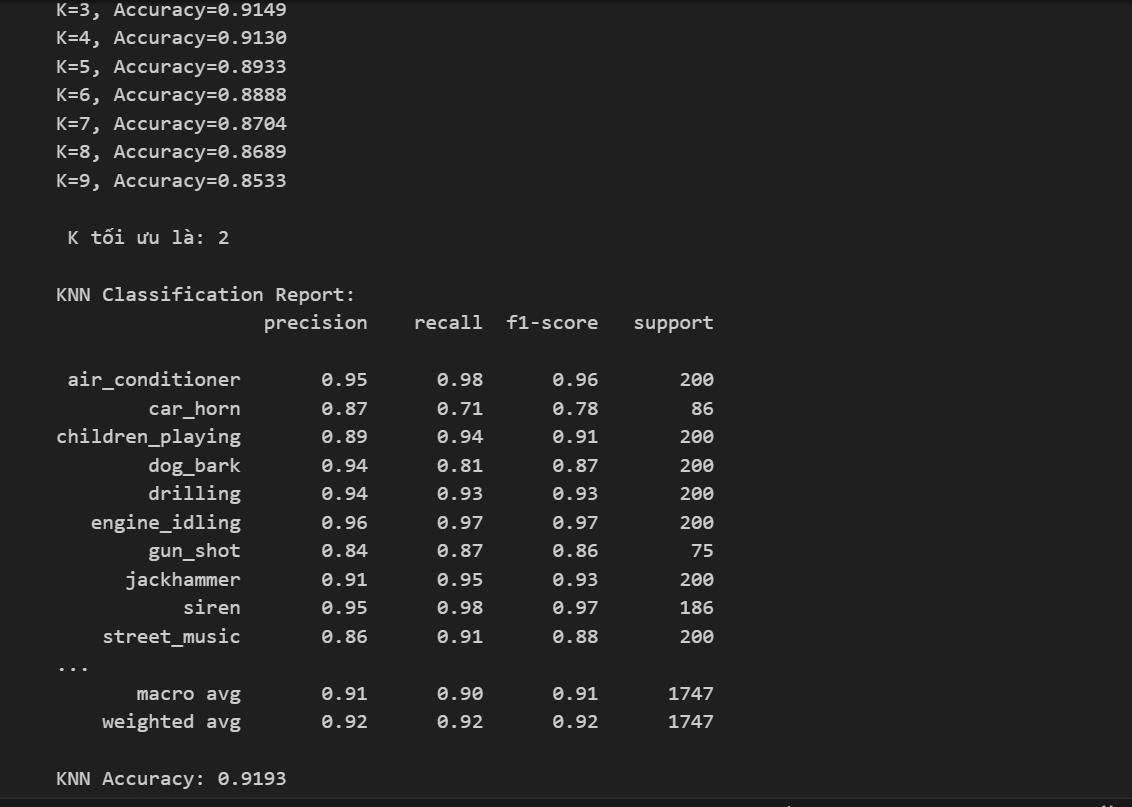
* Mục tiêu: Cài đặt thủ công mô hình KNN để phân loại dữ liệu mới bằng cách:
* Tính khoảng cách đến các mẫu huấn luyện.
* Chọn k hàng xóm gần nhất.
* Dự đoán nhãn qua bỏ phiếu đa số.

| # KNN thủ công dùng ma trận khoảng cách  class CustomKNN:  def \_\_init\_\_(self, k=5):  self.k = k  def fit(self, X, y):  self.X\_train = X  self.y\_train = y  def predict(self, X\_test):  distances = euclidean\_distance\_matrix(X\_test, self.X\_train)  predictions = []  for row in distances:  k\_idx = np.argsort(row)[:self.k]  k\_labels = self.y\_train[k\_idx]  most\_common = Counter(k\_labels).most\_common(1)[0][0]  predictions.append(most\_common)  return np.array(predictions) |
| --- |

**Bước 4: Huấn luyện và đánh giá mô hình KNN**

* Mục tiêu: Xây dựng, huấn luyện và đánh giá mô hình KNN thủ công với giá trị k tối ưu, nhằm phân loại chính xác âm thanh trên tập dữ liệu UrbanSound8K.

| k\_range = range(1, 10)  best\_k = find\_best\_k(features\_scaled, labels\_encoded, k\_range)  print(f"\n K tối ưu là: {best\_k}")  X\_train, X\_test, y\_train, y\_test = train\_test\_split(features\_scaled, labels\_encoded, test\_size=0.2, random\_state=42, stratify=labels\_encoded)  knn\_model = CustomKNN(k=best\_k)  knn\_model.fit(X\_train, y\_train)  y\_pred = knn\_model.predict(X\_test)  print("\nKNN Classification Report:")  print(classification\_report(y\_test, y\_pred, target\_names=label\_map.keys()))  print(f"KNN Accuracy: {accuracy\_score(y\_test, y\_pred):.4f}") |
| --- |

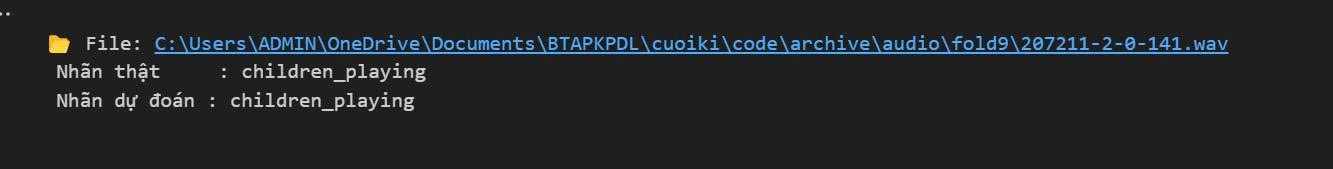


*Hình 17 : Kết quả đánh giá mô hình KNN với K tối ưu = 1*

* Mô hình KNN đạt độ chính xác cao (91.93%) với **k =1**. Hầu hết các lớp đều được phân loại tốt, nhưng một số lớp như **car\_horn** có recall thấp, cho thấy mô hình gặp khó khăn với các âm thanh ít mẫu hoặc dễ nhầm lẫn

**Bước 5: Dự Đoán file âm thanh mới từ nhập tên file & folder**

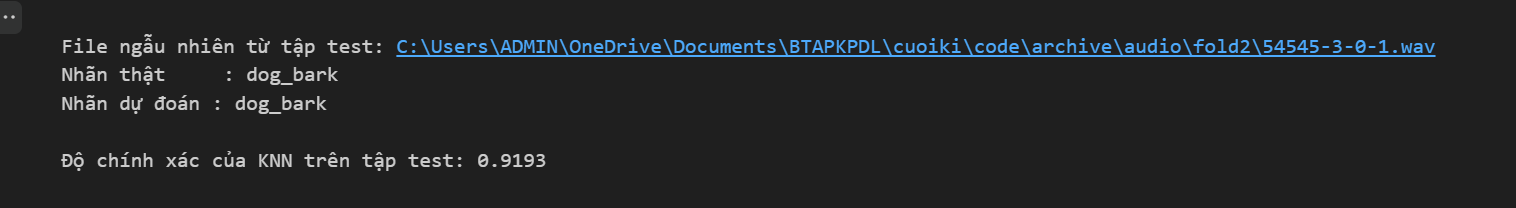
| def predict\_file(file\_path):  feat = extract\_audio\_features(file\_path)  if feat is None:  return None, None  feat\_scaled = (feat - mean) / std  pred\_knn\_encoded = knn\_model.predict([feat\_scaled])[0]  inv\_label\_map = {v: k for k, v in label\_map.items()}  return inv\_label\_map[pred\_knn\_encoded], None  selected\_filename = input("\nNhập tên file .wav (ví dụ: 54898-8-0-2.wav): ")  selected\_fold = input("Nhập số fold (ví dụ: 8): ")  example\_file = os.path.join(AUDIO\_BASE\_DIR, f"fold{selected\_fold}", selected\_filename)  df\_match = df[  (df['fold'].astype(str) == str(selected\_fold)) &  (df['slice\_file\_name'] == selected\_filename)  ]  print(f"\n File: {example\_file}")  if df\_match.empty:  print(" Không tìm thấy nhãn thật trong metadata.")  pred\_knn\_label, \_ = predict\_file(example\_file)  print(f" Nhãn dự đoán : {pred\_knn\_label}")  else:  selected\_label = df\_match['class'].values[0]  pred\_knn\_label, \_ = predict\_file(example\_file)  print(f" Nhãn thật : {selected\_label}")  print(f" Nhãn dự đoán : {pred\_knn\_label}") |
| --- |

****

*Hình 18 : Kết quả dự đoán âm thanh đầu vào bằng KNN*

**Bước 6: Dự đoán file âm thanh bên trong tập test**

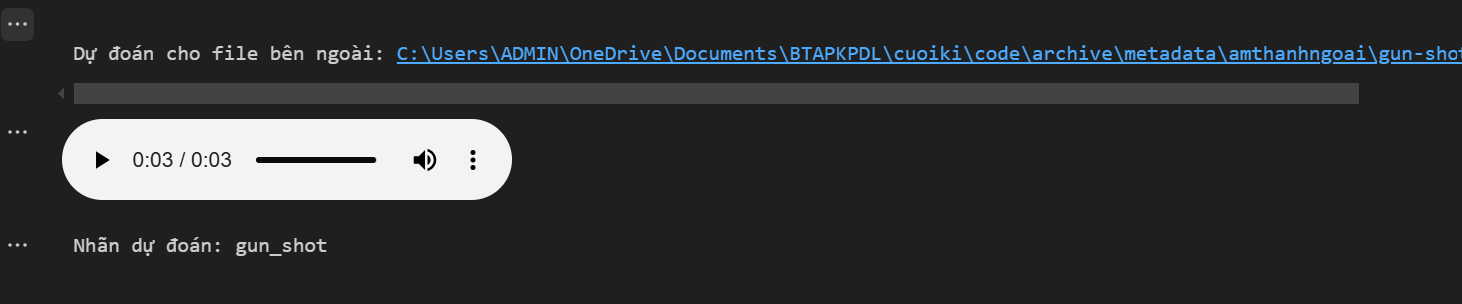
| # DỰ ĐOÁN 1 FILE NGẪU NHIÊN TỪ TẬP TEST  # Lấy index thực tế trong df\_exist tương ứng với tập test  def find\_original\_indices(X\_all, X\_subset):  return [i for i, x in enumerate(X\_all) if any((x == row).all() for row in X\_subset)]  # Tìm chỉ số gốc của tập test trong toàn bộ dữ liệu  test\_indices = find\_original\_indices(features\_scaled, X\_test)  # Chọn ngẫu nhiên 1 chỉ số từ tập test  random\_idx = random.choice(test\_indices)  # Lấy thông tin file từ metadata gốc  random\_row = df\_exist.iloc[random\_idx]  selected\_fold = random\_row['fold']  selected\_filename = random\_row['slice\_file\_name']  selected\_label = random\_row['class']  # Đường dẫn đầy đủ đến file âm thanh  example\_file = os.path.join(AUDIO\_BASE\_DIR, f"fold{selected\_fold}", selected\_filename)  # Hiển thị thông tin và dự đoán  print(f"\n File ngẫu nhiên từ tập test: {example\_file}")  print(f" Nhãn thật : {selected\_label}")  pred\_knn\_label, \_ = predict\_file(example\_file)  print(f" Nhãn dự đoán : {pred\_knn\_label}")  y\_test\_pred = knn\_model.predict(X\_test)  y\_test\_true = np.argmax(y\_test, axis=1) if len(y\_test.shape) > 1 else y\_test  # Tính độ chính xác  knn\_test\_acc = accuracy\_score(y\_test\_true, y\_test\_pred)  print(f"\n Độ chính xác của KNN trên tập test: {knn\_test\_acc:.4f}") |
| --- |

****

*Hình 19: Dự đoán file âm thanh từ tập test*

**Bước 7: Dự đoán file âm thanh bên ngoài tập dữ liệu**

| # Dự đoán file bên ngoài  def predict\_external\_file(file\_path):  if not os.path.isfile(file\_path):  print(" File không tồn tại.")  return  print(f"\n Dự đoán cho file bên ngoài: {file\_path}")  ipd.display(ipd.Audio(file\_path))  feat = extract\_audio\_features(file\_path)  if feat is None:  print(" Không thể trích xuất đặc trưng.")  return  feat\_scaled = (feat - mean) / std  pred\_encoded = knn\_model.predict([feat\_scaled])[0]  inv\_label\_map = {v: k for k, v in label\_map.items()}  predicted\_label = inv\_label\_map[pred\_encoded]  print(f" Nhãn dự đoán: {predicted\_label}")  # Nhập đường dẫn file ngoài  external\_file = input("\nNhập đường dẫn file .wav ngoài: ")  if external\_file.strip() != "":  predict\_external\_file(external\_file.strip()) |
| --- |

****

*Hình 20: Dự đoán file âm thanh ngoài*

## **Trực quan hóa kết quả dự đoán và đánh giá mô hình**

|  | Hình trên là ma trận nhầm lẫn của mô hình KNN, thể hiện mức độ chính xác trong việc phân loại từng loại âm thanh. Mỗi hàng biểu diễn nhãn thật và mỗi cột biểu diễn nhãn dự đoán. Các giá trị lớn tập trung trên đường chéo chính cho thấy phần lớn các âm thanh được phân loại đúng. Mô hình phân loại tốt với các lớp như **air\_conditioner, engine\_idling, siren và jackhammer**. Tuy nhiên, vẫn xảy ra nhầm lẫn ở một số lớp như **car\_horn, dog\_bark và gun\_shot**, cho thấy những loại âm thanh này dễ bị lẫn với các lớp khác. Ma trận này giúp đánh giá trực quan hiệu quả và hạn chế của mô hình. |
| --- | --- |
|  | Hình trên là bảng thống kê đúng/sai theo từng lớp âm thanh, cho biết độ chính xác (precision), độ bao phủ (recall), điểm F1, số mẫu phân loại đúng (**correct**) và sai **(incorrect)** của từng nhãn. Các lớp như **air\_conditioner, engine\_idling, siren và jackhammer** đạt độ chính xác và F1-score cao, đồng thời có số lượng mẫu sai rất ít. Ngược lại, **car\_horn và dog\_bark** có số lần dự đoán sai nhiều hơn, phản ánh mức độ khó trong việc phân biệt các âm thanh này. Bảng này giúp phân tích sâu hiệu quả mô hình trên từng loại âm thanh cụ thể. |
|  | Biểu đồ trên thể hiện Precision, Recall và F1-score của mô hình KNN theo từng lớp âm thanh. Các nhãn như **engine\_idling, siren, air\_conditione**r đạt chỉ số gần 1, cho thấy mô hình phân loại rất hiệu quả. Trong khi đó, **car\_horn và dog\_bark** có độ chính xác và F1 thấp hơn, phản ánh độ khó khi phân biệt các âm thanh này. Biểu đồ giúp trực quan hóa hiệu suất của mô hình trên từng loại âm thanh một cách rõ ràng. |
|  | Biểu đồ thể hiện số lượng mẫu phân loại đúng và sai theo từng lớp âm thanh của mô hình KNN. Dễ thấy các lớp như **air\_conditioner, engine\_idling, siren, jackhammer** có số mẫu đúng gần như tuyệt đối. Ngược lại, **car\_horn và dog\_bark** có tỷ lệ sai cao hơn đáng kể, cho thấy mô hình gặp khó khăn hơn khi phân loại các âm thanh này. |

# 

# 

# **Chương V. Xây dựng thuật toán Random Forest (RF)**

# 

## **Lý thuyết**

* Random Forest là một thuật toán học máy thuộc nhóm học có giám sát (supervised learning) và được sử dụng phổ biến trong các bài toán phân loại (classification) và hồi quy (regression). Thuật toán này là một dạng của tập hợp học (ensemble learning), nơi mà nhiều mô hình yếu (weak learners), cụ thể là các cây quyết định (decision trees), được kết hợp lại để tạo thành một mô hình mạnh mẽ hơn.



**\*Thuật ngữ “Random” trong Random Forest xuất phát từ hai yếu tố chính:**

**Ngẫu nhiên trong chọn mẫu**

* Thay vì sử dụng toàn bộ dữ liệu huấn luyện để xây dựng từng cây quyết định, thuật toán Random Forest chọn một mẫu ngẫu nhiên từ tập dữ liệu (với hoàn lại) để xây dựng mỗi cây. Kỹ thuật này được gọi là Bagging (Bootstrap Aggregating). Bagging giúp giảm thiểu phương sai của mô hình, cải thiện độ chính xác tổng thể.

**Ngẫu nhiên trong chọn đặc trưng**

* Khi tạo các nút trong mỗi cây, chỉ một tập con ngẫu nhiên của tất cả các đặc trưng được xem xét để chọn đặc trưng tốt nhất tại mỗi bước. Điều này giúp các cây quyết định đa dạng hơn, giảm thiểu hiện tượng overfitting và đảm bảo rằng các cây không bị phụ thuộc quá mức vào một đặc trưng cụ thể nào đó.

## **Các bước của thuật toán**

* Bước 1: Tạo các tập dữ liệu huấn luyện con.
* Từ tập dữ liệu gốc D, tạo ra n tập con bằng cách lấy mẫu ngẫu nhiên có hoàn lại (tức là 1 mẫu có thể xuất hiện nhiều lần)
* Mỗi tập con này có cùng kích thước với tập dữ liệu gốc(hoặc nhỏ hơn)
* Bước 2: Xây dựng nhiều cây quyết định.
* Với mỗi tập con, huấn luyện 1 cây quyết định nhưng có điều kiện đặc biệt sau:
* Tại mỗi nút chia, chỉ xét 1 tập con ngẫu nhiên của các đặc trưng, không xét toàn bộ.
* Điều này giúp các cây khác nhau nhiều hơn
* Mỗi cây được xây dựng đầy đủ (không cắt tỉa)
* Bước 3: Dự đoán bằng bỏ phiếu hoặc trung bình.
* Khi dự đoán mẫu mới:
* 1: Đưa mẫu vào từng cây trong rừng:
* 2: Với thuật toán phân loại:
* Mỗi cây cho ra 1 nhãn -> chọn nhãn được bỏ phiếu nhiều nhất
* 3: Với bài toán hồi quy:
* Trung bình giá trị đầu ra từ tất cả các cây

## **Xây dựng thuật toán**

1. Tìm ra cấu hình tốt nhất cho mô hình

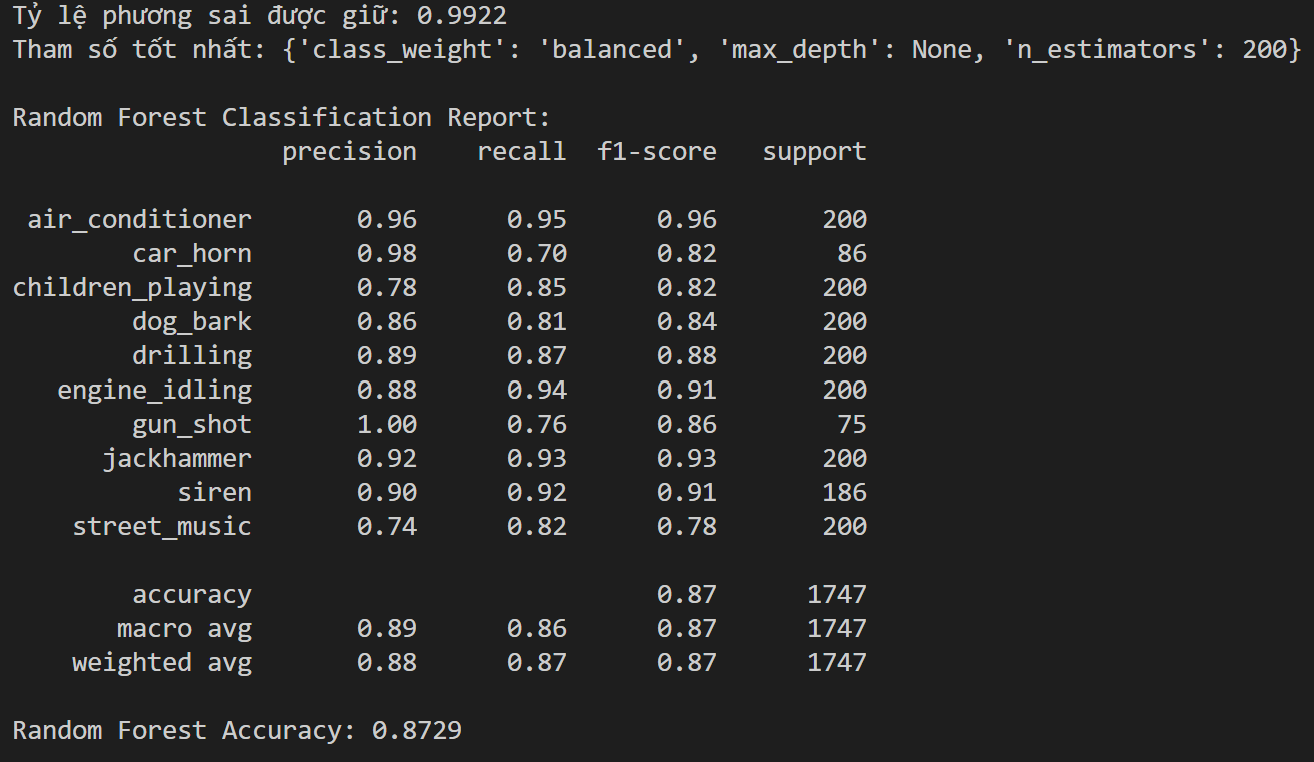
| def tune\_rf(X, y):  param\_grid = {  'n\_estimators': [100, 200],  'max\_depth': [10, 20, None],  'class\_weight': ['balanced']  }  rf = RandomForestClassifier(random\_state=42)  grid\_search = GridSearchCV(rf, param\_grid, cv=5, scoring='accuracy', n\_jobs=-1)  grid\_search.fit(X, y)  print(f"Tham số tốt nhất: {grid\_search.best\_params\_}")  return grid\_search.best\_estimator\_ |
| --- |

****

*Hình 21: Tham số tốt nhất*

1. Huấn luyện và đánh giá mô hình

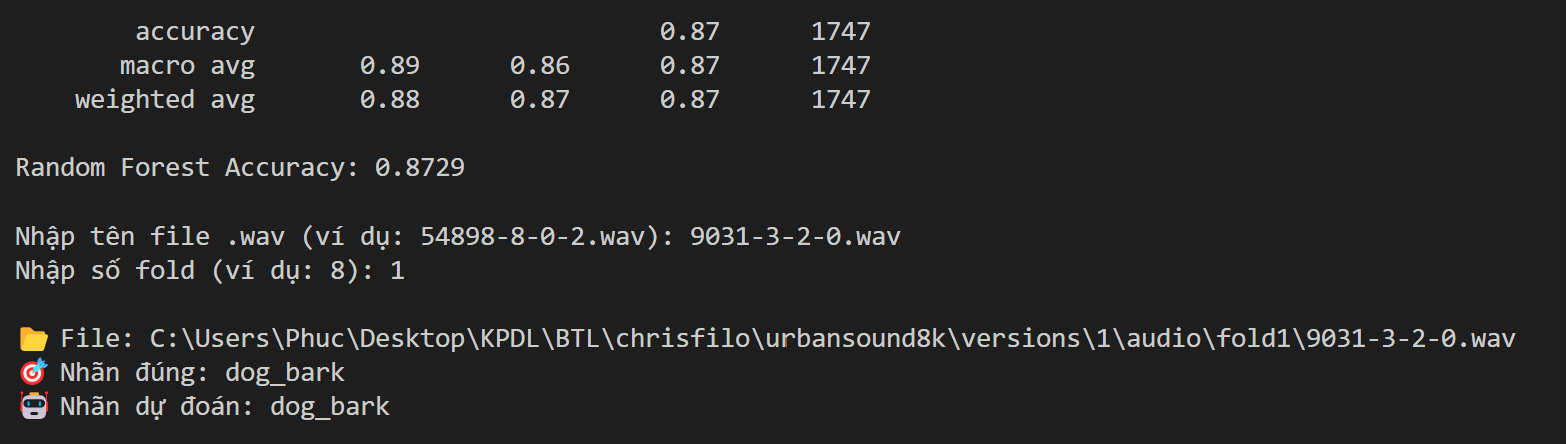
| # Huấn luyện và đánh giá  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  features\_pca, labels\_encoded, test\_size=0.2, random\_state=42, stratify=labels\_encoded  )  rf\_model = tune\_rf(X\_train, y\_train)  y\_pred = rf\_model.predict(X\_test)  print("\nRandom Forest Classification Report:")  print(classification\_report(y\_test, y\_pred, target\_names=label\_map.keys()))  print(f"Random Forest Accuracy: {accuracy\_score(y\_test, y\_pred):.4f}") |
| --- |



*Hình 22: Huấn luyện và đánh giá*

3:Dự đoán lớp âm thanh của 1 file cụ thể bằng mô hình Random Forest

| # Hàm dự đoán file audio  def predict\_file(file\_path):  feat = extract\_audio\_features(file\_path)  if feat is None:  return None  feat\_scaled = (feat - mean) / std  feat\_pca = pca.transform([feat\_scaled])[0]  pred = rf\_model.predict([feat\_pca])[0]  try:  return id\_to\_label[int(pred)] # Sử dụng id\_to\_label và ép kiểu về int  except KeyError:  print(f" Dự đoán {pred} không có trong id\_to\_label: {id\_to\_label}")  return None  # Nhập file để dự đoán  selected\_filename = input("\nNhập tên file .wav (ví dụ: 54898-8-0-2.wav): ")  selected\_fold = input("Nhập số fold (ví dụ: 8): ")  example\_file = os.path.join(AUDIO\_DIR, f'fold{selected\_fold}', selected\_filename)  df\_match = df[(df['fold'].astype(str) == str(selected\_fold)) & (df['slice\_file\_name'] == selected\_filename)]  print(f"\n File: {example\_file}")  if df\_match.empty:  print(" Không tìm thấy nhãn thật trong metadata.")  else:  true\_label = df\_match['class'].iloc[0]  print(f" Nhãn đúng: {true\_label}")  pred\_label = predict\_file(example\_file)  print(f" Nhãn dự đoán: {pred\_label}") |
| --- |



*Hình 23. Kết quả dự đoán Random Forest*

## 

## **Trực quan hóa và đánh giá chi tiết hiệu suất mô hình Random Forest bằng các biểu đồ và số liệu**

|  | **Ý nghĩa của hình:**  **Trục dọc (Thực tế)**: là **nhãn thực sự** của các đoạn âm thanh.  **Trục ngang (Dự đoán)**: là **nhãn mà mô hình dự đoán** cho các đoạn âm thanh đó.  **Các ô vuông**: biểu diễn **số lượng mẫu** được phân loại theo từng cặp (thực tế, dự đoán).  **Màu xanh đậm**: biểu thị số lượng lớn, thể hiện mô hình tự tin và chính xác ở nhãn đó.  **Màu nhạt hoặc trắng**: số lượng ít, hoặc không có nhầm lẫn. |
| --- | --- |
|  | Precision: Độ chính xác: Trong tất cả các dự đoán là lớp này, có bao nhiêu là đúng  recall: Độ bao phủ: Trong tất cả các mẫu thực sự thuộc lớp này, mô hình dự đoán đúng bao nhiêu  f1-score: Trung bình hài hòa giữa precision và recall  support: Số lượng mẫu thực sự thuộc lớp đó trong tập test  accuracy:Tỷ lệ dự đoán đúng tổng thể: 0.8729 → tức là mô hình dự đoán đúng **87.29%**  macro avg:Trung bình không trọng số (tính trung bình từng lớp)  weighted avg:Trung bình có trọng số theo số lượng mẫu mỗi lớp |
|  | Biểu đồ thể hiện Precision, Recall và F1-score của mô hình RF theo từng lớp âm thanh.  - Các lớp **air\_conditioner** ,**jackhammer và siren** đạt cả 3 chỉ số gần 1, cho thấy mô hình phân loại rất hiệu quả.  - Các lớp còn lại có sự cân bằng giữa precision và recall, nhưng car\_horn và gunshot cần cải thiện recall để tăng f1-score. Điều này có thể liên quan đến đặc trưng của dữ liệu hoặc dữ liệu bị mất cân bằng.  - Recall thấp ở một số lớp cho thấy mô hình bỏ sót mẫu. |
|  | Biểu đồ Feature Importance thể hiện tầm quan trọng của 20 thành phần chính (PC1-PC20) sau PCA.  - Từ PC1 đến PC5: có tầm quan trọng cao (~0.06- 0.08), cho thấy 5 thành phần đầu tiên đóng góp lớn vào mô hình.  - Còn từ PC6-PC20: giảm dần, với PC10-PC20 gần 0.02-0.03, cho thấy lượng thông tin giảm dần.  - Các thành phần có sự phân bố đều ( không có lớp nào quá thấp ) cho thấy dữ liệu ban đầu (MFCC, Chroma, ZCR, RMS) có lượng thông tin bao phủ rộng. |
|  | Biểu đồ thể hiện số lượng mẫu phân loại đúng và sai theo từng lớp âm thanh của mô hình Random Forest.  Dễ thấy các lớp như **air\_conditioner, engine\_idling, siren, jackhammer** có số mẫu đúng gần như tuyệt đối. Ngược lại, **car\_horn và dog\_bark và street\_music** có tỷ lệ sai cao hơn đáng kể, cho thấy mô hình gặp khó khăn hơn khi phân loại các âm thanh này. |

## 

# **Chương VI. Xây dựng thuật toán CNN**

## **Lý thuyết**

* CNN (Convolutional Neural Network) là một mạng nơ-ron chuyên biệt, mạnh trong việc xử lý dữ liệu dạng ảnh hoặc tín hiệu có cấu trúc không gian (như ảnh, sóng âm…). CNN sử dụng các lớp tích chập kích hoạt (activation), và gộp (pooling) để tự động trích xuất đặc trưng không gian từ dữ liệu đầu vào. Sau các lớp convolution là các lớp fully connected giúp thực hiện phân loại.
* Mục đích: Áp dụng CNN để phân loại âm thanh thành một trong 10 lớp như "dog\_bark", "siren", "drilling".. từ ảnh đặc trưng Mel-spectrogram đã trích xuất từ tín hiệu âm thanh. Mục tiêu là xác định loại âm thanh đầu vào một cách chính xác.
* Input:

+ Ảnh Mel-spectrogram đã chuẩn hóa từ các file âm thanh (.wav).

+ Nhãn tương ứng với mỗi mẫu (ở dạng số hoặc chuỗi).

+ Một file âm thanh mới cần dự đoán.

* Output: Nhãn dự đoán cho âm thanh đầu vào (dạng số hoặc dạng chuỗi: "car\_horn", "engine\_idling",...).

## **Các bước của thuật toán**

* Bước 1: Nhập dữ liệu đầu vào**:**
* Dữ liệu có thể là ảnh (RGB hoặc grayscale) hoặc biểu diễn dạng ảnh như Mel-spectrogram (đối với âm thanh).
* Thường chuẩn hóa dữ liệu về khoảng [0,1].
* Bước 2: Lớp Convolution (Tích chập)
* Áp dụng các bộ lọc (kernel) trượt trên ảnh để trích xuất các đặc trưng cục bộ như cạnh, góc, kết cấu,...
* Kết quả là các feature map thể hiện đặc trưng.
* Bước 3: Hàm kích hoạt (ReLU)
* Áp dụng hàm phi tuyến ReLU: **f(x) = max(0, x)** để giữ lại thông tin dương và loại bỏ âm.
* Giúp mạng học được các đặc trưng phức tạp hơn.
* Bước 4: Lớp Pooling (Gộp)
* Thường dùng MaxPooling để chọn giá trị lớn nhất trong từng vùng nhỏ.
* Mục đích: giảm kích thước dữ liệu và tính toán, tăng khả năng tổng quát.
* Bước 5: (Lặp lại bước 2–4)
* Có thể thêm nhiều tầng convolution + ReLU + pooling để học các đặc trưng sâu hơn.
* Bước 6: Làm phẳng (Flatten)
* Chuyển các feature maps 2D thành 1 vector 1D để đưa vào lớp fully connected.
* Bước 7: Lớp Fully Connected (Dense)
* Mô hình hóa mối quan hệ giữa các đặc trưng đã trích xuất.
* Kết nối toàn bộ để học cách phân loại.
* Bước 8: Lớp đầu ra (Output Layer)
* Dùng Softmax nếu phân loại nhiều lớp, Sigmoid nếu phân loại nhị phân.
* Trả về xác suất dự đoán mỗi lớp.
* Bước 9: Tính hàm mất mát (Loss function)
* Dùng Cross-Entropy để tính sai lệch giữa dự đoán và nhãn thật.
* Loss càng nhỏ chứng tỏ mô hình học càng tốt.
* Bước 10: Lan truyền ngược (Backpropagation) và cập nhật trọng số
* Tính gradient và cập nhật các trọng số trong mạng bằng thuật toán tối ưu (như Adam hoặc SGD).
* Quá trình lặp lại qua nhiều epoch cho đến khi hội tụ.

## **Xây dựng thuật toán**

**Bước 1: Chuẩn bị và trích xuất đặc trưng từ âm thanh (Tiền xử lí dữ liệu)**

1. Xây dựng hàm Mel-spectrogram để chuyển âm thanh thành đặc trưng ảnh

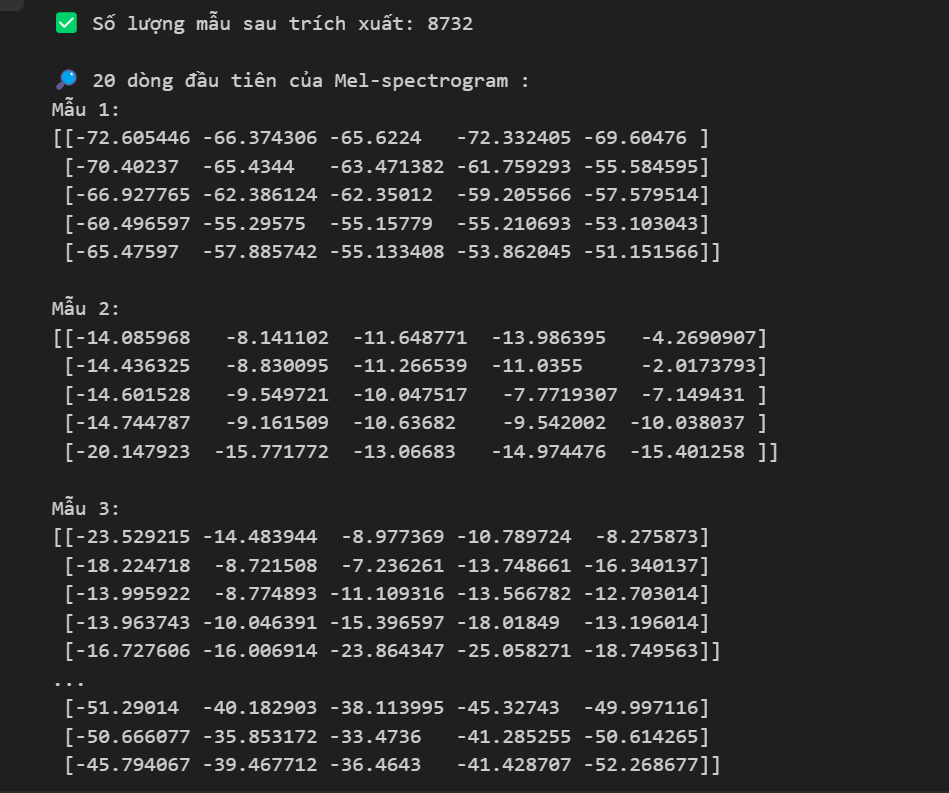
* Mục tiêu: Chuyển âm thanh → đặc trưng ảnh (để dùng CNN như xử lý ảnh)
* Đọc metadata .csv để biết tên file, nhãn.
* Duyệt qua từng file âm thanh .wav:  
  + Tải file bằng librosa.
  + Chuyển thành Mel-spectrogram (ảnh thể hiện năng lượng theo thời gian và tần số).
  + Cắt/padding để ảnh có cùng kích thước (128 x 174).
  + Lưu lại ảnh (ma trận) + nhãn (label).

| # TRÍCH XUẤT MEL-SPECTROGRAM  def extract\_mel\_spectrogram(file\_path, n\_mels=128, max\_len=174):  try:  y, sr = librosa.load(file\_path, sr=22050)  mel = librosa.feature.melspectrogram(y=y, sr=sr, n\_mels=n\_mels)  log\_mel = librosa.power\_to\_db(mel, ref=np.max)  if log\_mel.shape[1] < max\_len:  pad\_width = max\_len - log\_mel.shape[1]  log\_mel = np.pad(log\_mel, pad\_width=((0, 0), (0, pad\_width)), mode='constant')  else:  log\_mel = log\_mel[:, :max\_len]  return log\_mel  except:  return None |
| --- |

1. Encode và chuẩn bị dữ liệu huấn luyện

* Mục tiêu: Biến nhãn thành số, chia dữ liệu
* Sử dụng LabelEncoder để biến tên lớp (vd: "car\_horn") → số nguyên.
* One-hot encode để phục vụ cho phân loại nhiều lớp (multi-class).
* Tách dữ liệu thành tập train/test (thường 80/20) với stratify.

| # XỬ LÝ TOÀN BỘ DỮ LIỆU  X, y, paths = [], [], []  for \_, row in df.iterrows():  fold = f"fold{row['fold']}"  file\_path = os.path.join(AUDIO\_DIR, fold, row['slice\_file\_name'])  mel\_spec = extract\_mel\_spectrogram(file\_path)  if mel\_spec is not None:  X.append(mel\_spec)  y.append(row['class'])  paths.append(file\_path)  X = np.array(X)[..., np.newaxis]  le = LabelEncoder()  y\_encoded = le.fit\_transform(y)  y\_onehot = tf.keras.utils.to\_categorical(y\_encoded)  print(f"\n Số lượng mẫu sau trích xuất: {len(X)}")  print("\n 20 dòng đầu tiên của Mel-spectrogram :")  for i in range(min(20, len(X))):  print(f"Mẫu {i + 1}:\n{X[i][:5, :5, 0]}\n")  # TÁCH TRAIN / TEST  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_onehot, test\_size=0.2, stratify=y\_onehot, random\_state=42) |
| --- |

****

*Hình 24 : Mel-spectrogram đầu vào CNN – 20 mẫu đầu tiên*

**Bước 2: Trực quan hóa dữ liệu**

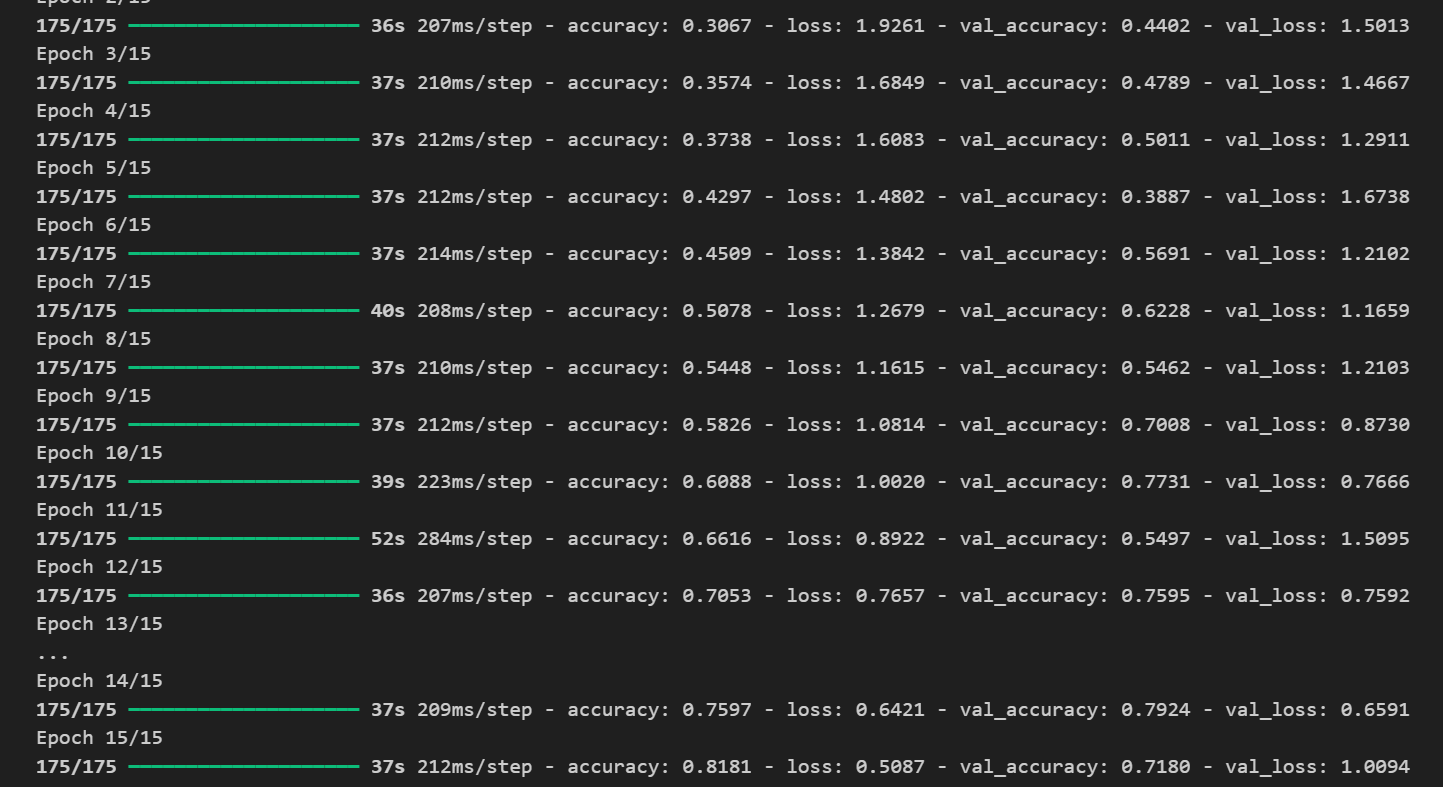
|  | Hình ảnh trên là **Mel-spectrogram đại diện cho từng lớp âm thanh trong tập dữ liệu UrbanSound8K**, trong đó mỗi hàng thể hiện một lớp nhãn âm thanh như **dog\_bark, car\_horn, siren, street\_music,**.... Mỗi ảnh trong hàng là một Mel-spectrogram – biểu diễn phổ tần của tín hiệu âm thanh theo thời gian, với trục tung là tần số theo thang Mel, trục hoành là thời gian, và màu sắc thể hiện cường độ năng lượng âm thanh (từ tối đến sáng). Một số ảnh có phần nền màu vàng do âm thanh ngắn hơn độ dài chuẩn (174 frame) nên được **pad thêm** để đồng nhất kích thước đầu vào. Các Mel-spectrogram này cho thấy đặc trưng riêng của từng loại âm thanh: ví dụ, **air\_conditioner, engine\_idling, siren có dạng phổ đều đặn, kéo dài; gun\_shot và dog\_bark có đặc trưng ngắn, đột ngột; trong khi children\_playing hay street\_music** có phổ biến thiên phức tạp. Những đặc trưng này là cơ sở để các mô hình học sâu, đặc biệt là mạng nơ-ron tích chập (CNN), khai thác nhằm phân loại chính xác các loại âm thanh khác nha |
| --- | --- |
|  | Hình ảnh trên là biểu đồ t-SNE trực quan hóa các mẫu âm thanh trong UrbanSound8K sau khi trích xuất đặc trưng Mel-spectrogram và giảm chiều. Mỗi điểm đại diện cho một mẫu, được tô màu theo nhãn lớp. Một số lớp như “jackhammer”, “air\_conditioner”, “dog\_bark” hay “drilling” tách biệt rõ, cho thấy đặc trưng dễ phân biệt. Ngược lại, các lớp như “children\_playing” hay “street\_music” bị chồng lấn, phản ánh sự tương đồng đặc trưng. Biểu đồ giúp đánh giá mức độ phân biệt giữa các loại âm thanh trước khi huấn luyện mô hình. |

# 

**Bước 3: Xây dựng mô hình và huấn luyện mô hình CNN**

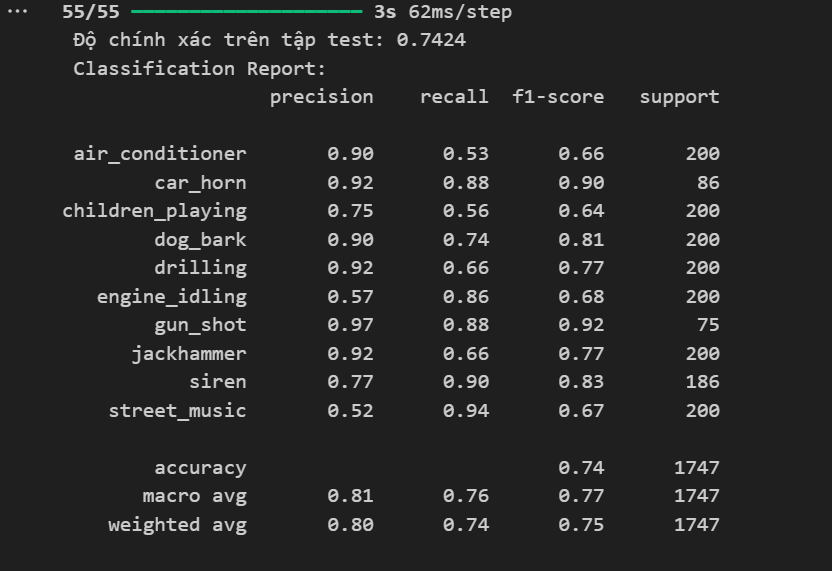
* Mục tiêu: Thiết kế mô hình học từ ảnh Mel để dự đoán nhãn
* Conv2D: Lọc đặc trưng từ ảnh đầu vào bằng các kernel 3x3.
* ReLU: Kích hoạt phi tuyến giúp mô hình học tốt hơn.
* MaxPooling: Giảm chiều ảnh (rút trích đặc trưng).
* BatchNormalization: Ổn định gradient, tăng tốc huấn luyện.
* Flatten: Biến ma trận đặc trưng thành vector.
* Dense + Dropout: Mạng fully-connected cuối cùng → phân loại.
* Softmax: Lớp đầu ra với xác suất cho từng lớp.

| # MÔ HÌNH CNN  model = models.Sequential([  layers.Conv2D(32, (3, 3), activation='relu', input\_shape=X\_train.shape[1:]),  layers.MaxPooling2D((2, 2)),  layers.BatchNormalization(),  layers.Conv2D(64, (3, 3), activation='relu'),  layers.MaxPooling2D((2, 2)),  layers.BatchNormalization(),  layers.Conv2D(128, (3, 3), activation='relu'),  layers.MaxPooling2D((2, 2)),  layers.Flatten(),  layers.Dense(128, activation='relu'),  layers.Dropout(0.3),  layers.Dense(y\_train.shape[1], activation='softmax')  ])  model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  # ======== HUẤN LUYỆN MÔ HÌNH ========  history = model.fit(X\_train, y\_train, epochs=15, batch\_size=32, validation\_split=0.2) |
| --- |



*Hình 25 : Tiến trình huấn luyện mô hình CNN qua các epoch*

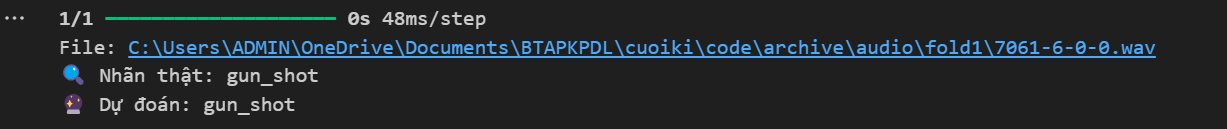
| # Đánh giá mô hình  test\_loss, test\_acc = model.evaluate(X\_test, y\_test, verbose=0)  y\_pred = model.predict(X\_test)  y\_pred\_labels = np.argmax(y\_pred, axis=1)  y\_true\_labels = np.argmax(y\_test, axis=1)  print(f" Độ chính xác trên tập test: {test\_acc:.4f}")  print(" Classification Report:")  print(classification\_report(y\_true\_labels, y\_pred\_labels, target\_names=le.classes\_))  test\_size=0.2, stratify=y\_onehot, random\_state=42) |
| --- |



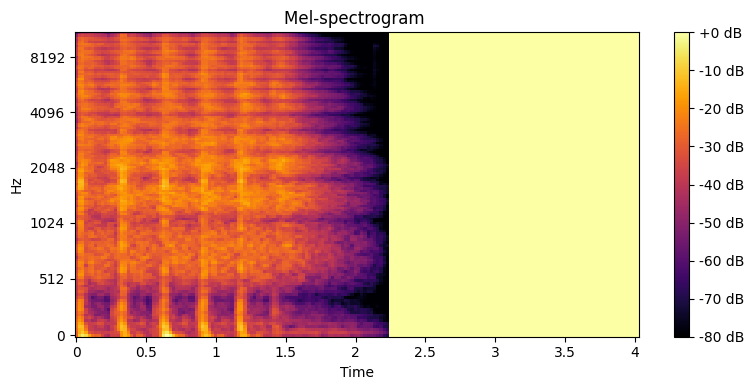
*Hình 26 : Classification Report – Kết quả mô hình CNN*

**Bước 4: Dự đoán một file**

| def predict\_from\_fold\_and\_file(fold\_name, file\_name):  file\_path = os.path.join(AUDIO\_DIR, fold\_name, file\_name)  if not os.path.exists(file\_path):  print("\u274c File không tồn tại.")  return  mel = extract\_mel\_spectrogram(file\_path)  mel\_input = mel[np.newaxis, ..., np.newaxis]  prediction = model.predict(mel\_input)  predicted\_label = le.inverse\_transform([np.argmax(prediction)])[0]  # Lấy nhãn thật từ DataFrame  actual\_row = df[(df['fold'] == int(fold\_name.replace('fold', ''))) & (df['slice\_file\_name'] == file\_name)]  actual\_label = actual\_row['class'].values[0] if not actual\_row.empty else "Không rõ"  print(f"File: {file\_path}")  print(f" Nhãn thật: {actual\_label}")  print(f" Dự đoán: {predicted\_label}")  plt.figure(figsize=(8, 4))  librosa.display.specshow(mel, sr=22050, hop\_length=512, x\_axis='time', y\_axis='mel', cmap='inferno')  plt.title("Mel-spectrogram ")  plt.colorbar(format='%+2.0f dB')  plt.tight\_layout()  plt.show()  predict\_from\_fold\_and\_file("fold1", "7061-6-0-0.wav") |
| --- |

****

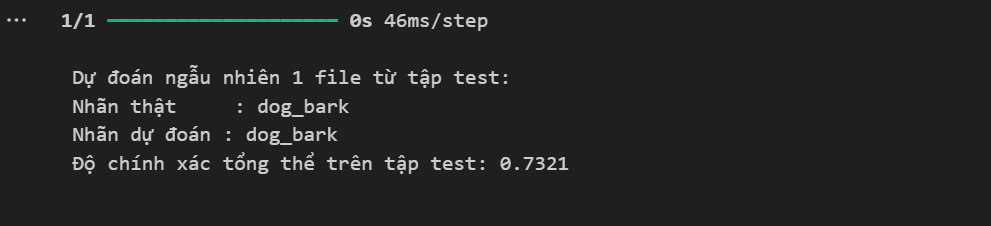
*Hình 27 : Dự đoán file bằng thuật toán CNN*



*Hình 28 : Mel-spectrogram mẫu âm thanh*

**Bước 5: Dự đoán một file từ tập test**

| # ======== DỰ ĐOÁN NGẪU NHIÊN MỘT FILE TỪ TẬP TEST ========  # Lấy chỉ số ngẫu nhiên trong tập test  rand\_idx = random.randint(0, len(X\_test) - 1)  # Dữ liệu và nhãn thật  sample = X\_test[rand\_idx][np.newaxis, ...] # thêm batch dimension  true\_label = le.inverse\_transform([np.argmax(y\_test[rand\_idx])])[0]  # Dự đoán  pred\_probs = model.predict(sample)  pred\_label = le.inverse\_transform([np.argmax(pred\_probs)])[0]  print("\n Dự đoán ngẫu nhiên 1 file từ tập test:")  print(f" Nhãn thật : {true\_label}")  print(f" Nhãn dự đoán : {pred\_label}")  print(f" Độ chính xác tổng thể trên tập test: {test\_acc:.4f}") |
| --- |

****

*Hình 29 : Dự đoán từ file từ tập test*

1. **Trực quan hóa kết quả dự đoán và đánh giá mô hình**

|  | Hình trên thể hiện biểu đồ độ chính xác (Accuracy) và hàm mất mát (Loss) của mô hình trong quá trình huấn luyện. Biểu đồ bên trái cho thấy độ chính xác trên tập huấn luyện (Train Acc) tăng đều theo số epoch, trong khi độ chính xác trên tập kiểm tra (Val Acc) dao động mạnh, cho thấy mô hình có dấu hiệu overfitting. Biểu đồ bên phải thể hiện giá trị hàm mất mát giảm đều trên tập huấn luyện (Train Loss), trong khi trên tập kiểm tra (Val Loss) lại dao động lớn và không ổn định, càng khẳng định hiện tượng overfitting xảy ra khi mô hình học tốt trên dữ liệu huấn luyện nhưng không tổng quát tốt trên dữ liệu mới. |
| --- | --- |
|  | Hình trên là **ma trận nhầm lẫn** cho mô hình CNN trên tập UrbanSound8K. Mô hình dự đoán tốt các lớp như **street\_music, siren, và engine\_idling, nhưng còn nhầm lẫn nhiều ở các lớp air\_conditioner và children\_playing**. Kết quả cho thấy mô hình nhận diện tốt các âm thanh đặc trưng, nhưng vẫn gặp khó khăn với các lớp dễ gây nhiễu. |
|  | Hình trên là **bảng thống kê độ chính xác theo từng lớp**. Các lớp như gun\_shot, car\_horn, và siren có **f1-score cao**, cho thấy mô hình phân loại tốt. Ngược lại, các lớp như air\_conditioner, children\_playing, và street\_music có **tỷ lệ sai cao**, phản ánh độ khó khi phân biệt các âm thanh này. Đây là cơ sở để điều chỉnh mô hình hoặc xử lý dữ liệu nâng cao.rõ ràng. |
|  | Biểu đồ trên thể hiện các chỉ số **Precision**, **Recall** và **F1-score** của mô hình **CNN** đối với từng lớp âm thanh khác nhau. Mỗi nhóm âm thanh như air\_conditioner, car\_horn, dog\_bark, .... được đánh giá dựa trên ba tiêu chí chính, nhằm phản ánh độ hiệu quả của mô hình trong việc phân loại. Kết quả cho thấy mô hình hoạt động tốt nhất với các lớp như gun\_shot, car\_horn, và drilling khi cả ba chỉ số đều đạt giá trị cao, gần 1. Trong khi đó, một số lớp như engine\_idling hay jackhammer có chỉ số thấp hơn, cho thấy mô hình gặp khó khăn hơn trong việc nhận diện chính xác các âm thanh này. Nhìn chung, biểu đồ cung cấp cái nhìn tổng quát về hiệu suất của mô hình CNN theo từng loại âm thanh. |
|  | Biểu đồ trên thể hiện **số lượng mẫu dự đoán đúng (correct)** và **sai (incorrect)** của mô hình **CNN** đối với từng lớp âm thanh. Trục hoành biểu diễn các lớp âm thanh như air\_conditioner, dog\_bark, siren, ..., trong khi trục tung thể hiện số lượng mẫu. Màu xanh nhạt biểu thị số mẫu được mô hình phân loại đúng, còn màu nâu biểu thị số mẫu bị phân loại sai. Có thể thấy một số lớp như gun\_shot, siren và street\_music có tỷ lệ phân loại đúng rất cao, với rất ít lỗi. Ngược lại, các lớp như children\_playing, jackhammer và dog\_bark có số lượng mẫu bị phân loại sai đáng kể, cho thấy mô hình gặp khó khăn hơn trong việc nhận diện các âm thanh này. Biểu đồ cung cấp cái nhìn trực quan về mức độ chính xác theo từng lớp và là cơ sở để đánh giá hiệu quả tổng thể của mô hình. |

# 

# **Chương VII. Kết luận và hướng phát triển**

## **Phân tích và đánh giá kết quả**

### **1.1. KNN (K-Nearest Neighbors)**

* Độ chính xác (Accuracy): 91.93% – cao nhất trong ba mô hình.
* F1-score trung bình: 0.91 (macro), 0.92 (weighted).
* Nhận xét:
* Mô hình đạt hiệu quả cao và ổn định trên hầu hết các lớp.
* Các lớp như air\_conditioner, engine\_idling, siren được phân loại rất tốt (f1-score ≈ 0.97).
* Lớp car\_horn có kết quả thấp nhất (f1-score: 0.78) do số lượng mẫu ít.
* Phù hợp cho tập dữ liệu có độ phân bố đồng đều.

### **1.2. Random Forest**

* Độ chính xác: 87.29%.
* F1-score trung bình: 0.87.
* Nhận xét:
* Cho kết quả khá tốt, đặc biệt ở các lớp gun\_shot (precision: 1.0), jackhammer, engine\_idling.
* Tuy nhiên, mô hình bị giảm hiệu quả ở các lớp như street\_music (f1-score: 0.78) và children\_playing.
* Ưu điểm là xử lý tốt dữ liệu mất cân bằng với class\_weight='balanced'.

### **1.3. CNN (Convolutional Neural Network)**

* Độ chính xác: 74.24% – thấp nhất trong 3 mô hình.
* F1-score trung bình: 0.77.
* Nhận xét:
* Mô hình có khả năng phân biệt tốt với các lớp như gun\_shot, car\_horn, siren.
* Tuy nhiên, nhiều lớp bị phân loại kém như air\_conditioner, children\_playing, street\_music (precision thấp).
* Có thể do mô hình chưa được tối ưu tốt hoặc chưa đủ dữ liệu huấn luyện.

## **Đánh giá mô hình**

* Trong quá trình thực nghiệm, ba mô hình học máy bao gồm **K-Nearest Neighbors (KNN), Random Forest,** và **Convolutional Neural Network (CNN)** đã được huấn luyện và đánh giá trên tập dữ liệu âm thanh nhằm phân loại 10 lớp khác nhau. Kết quả đánh giá được so sánh dựa trên các chỉ số như **độ chính xác (accuracy), precision, recall**, và **F1-score**.
* **Mô hình KNN** đạt kết quả tốt nhất với độ chính xác lên đến **91.93%** khi sử dụng giá trị K tối ưu là 2. Các chỉ số trung bình như macro **F1-score** và **weighted F1-score** lần lượt là **0.91** và **0.92**, thể hiện khả năng phân loại đồng đều giữa các lớp. Đặc biệt, các lớp như **air\_conditioner, engine\_idling** và **siren** đều có F1-score cao, trên **0.95**. Tuy nhiên, một số lớp như **car\_horn** và **gun\_shot** có recall thấp hơn, cho thấy mô hình vẫn gặp khó khăn trong việc nhận diện một số loại âm thanh ngắn hoặc hiếm gặp.
* **Mô hình Random Fores**t tuy có độ chính xác tổng thể **thấp hơn KNN (87.29%)**, nhưng vẫn thể hiện hiệu quả khá tốt với macro và weighted F1-score đều đạt **0.87**. Một điểm mạnh của Random Forest là khả năng xử lý dữ liệu không cân bằng, đặc biệt nhờ việc sử dụng tham số class\_weight='balanced'. Lớp gun\_shot có precision đạt tuyệt đối (1.00), tuy recall chỉ ở mức 0.76. Trong khi đó, các lớp như children\_playing và street\_music lại có F1-score thấp hơn, cho thấy mô hình còn hạn chế với các âm thanh có tính chất phức tạp và biến động cao.
* **Mô hình CNN** cho kết quả thấp nhất với độ chính xác chỉ đạt **74.24%**, cùng với **macro F1-score là 0.77**. Mặc dù mô hình học sâu như CNN thường có tiềm năng lớn trong xử lý tín hiệu âm thanh, song kết quả này cho thấy mô hình chưa được tối ưu hóa. Một số lớp như **air\_conditioner** và **engine\_idling** có recall khá thấp (lần lượt là 0.53 và 0.86), dẫn đến **F1-score không cao**. Ngược lại, các lớp như gun\_shot và car\_horn vẫn đạt F1-score tương đối cao (trên 0.90), phản ánh khả năng học đặc trưng mạnh của CNN với các loại âm thanh rõ ràng.
* Tổng kết lại, mô hình KNN cho hiệu quả phân loại tốt nhất trên tập dữ liệu hiện tại, theo sau là Random Forest và CNN. Tuy nhiên, với việc tinh chỉnh thêm kiến trúc mạng, tăng kích thước và chất lượng tập dữ liệu, mô hình CNN vẫn có tiềm năng vượt trội trong các ứng dụng thực tế đòi hỏi khả năng trích xuất đặc trưng sâu từ dữ liệu âm thanh.

## **Hướng phát triển**

* Để cải thiện hiệu quả phân loại âm thanh và mở rộng ứng dụng trong thực tế, một số hướng phát triển tiềm năng bao gồm:
* Mở rộng và cân bằng tập dữ liệu: Bổ sung thêm mẫu cho các lớp có dữ liệu ít hoặc thường bị phân loại sai như car\_horn, children\_playing, từ đó giúp mô hình học tốt hơn và giảm hiện tượng mất cân bằng.
* Tối ưu hóa mô hình học sâu: Cải thiện kiến trúc CNN bằng cách tăng số lớp, thêm Dropout, Batch Normalization hoặc sử dụng kiến trúc hiện đại như ResNet, MobileNet để tăng khả năng trích xuất đặc trưng mà vẫn đảm bảo hiệu suất tính toán.
* Kết hợp mô hình (ensemble learning): Áp dụng các kỹ thuật như bagging, boosting hoặc stacking để kết hợp nhiều mô hình, tận dụng ưu điểm của từng thuật toán và tăng độ chính xác tổng thể.
* Trích xuất đặc trưng nâng cao: Thay vì chỉ dùng Mel Spectrogram, có thể sử dụng các đặc trưng mạnh hơn như MFCC, chroma features, spectral contrast, hoặc thậm chí dùng học đặc trưng tự động với autoencoder.
* Sử dụng các phương pháp tiên tiến khác:
* Transfer Learning: Sử dụng mô hình tiền huấn luyện chuyên biệt cho âm thanh như YAMNet, VGGish, hoặc PANNs để tận dụng kiến thức từ các tập dữ liệu lớn.
* Attention Mechanism: Tích hợp cơ chế attention giúp mô hình tập trung vào vùng âm thanh quan trọng hơn trong chuỗi thời gian.
* Transformer cho âm thanh: Áp dụng mô hình transformer hoặc Audio Spectrogram Transformer (AST) – hiện đang được sử dụng hiệu quả trong nhiều bài toán nhận dạng âm thanh hiện đại.
* Triển khai ứng dụng thực tế: Phát triển hệ thống giám sát âm thanh theo thời gian thực (ví dụ: cảnh báo còi xe, phát hiện tiếng súng, âm thanh bất thường), ứng dụng trong an ninh, giao thông hoặc nhà thông minh.

## 