دانشکده: مهندسی مکانیک

پایانترم/پروژه درس هوش مصنوعی

موضوع:

پروژه تشخیص جنس قطعه با ضربه

استاد درس:

دکتر ذبیحی فر

گروه:

امیرحسین جامه بزرگ	۴۰۰۵۴۱۲۷۹
محمدحسین ابراهیمی	۴۰۰۵۴۱۰۰۹
سبحان مشهدی	۴۰۰۵۴۲۴۹۹

پاییز ۱۴۰۳

فهرست:

فهرست شکل‌ها:

مقدمه

مقدمه:

در دنیای امروز، استفاده از فناوری‌های نوین به‌ویژه هوش مصنوعی (AI) و یادگیری ماشین (ML) برای حل مسائل پیچیده و خودکارسازی فرآیندها به‌طور فزاینده‌ای گسترش یافته است. یکی از کاربردهای جذاب و نوظهور این فناوری‌ها، شناسایی جنس مواد مختلف از طریق تحلیل داده‌های صوتی است. صدای ضربه زدن به مواد مختلف، به دلیل تفاوت در ساختار فیزیکی و خواص مکانیکی آن‌ها، الگوهای منحصربه‌فردی ایجاد می‌کند که قابل تحلیل و طبقه‌بندی هستند.

این پروژه با هدف شناسایی جنس مواد مختلف نظیر چوب، پلاستیک، فلز، شیشه و ... از طریق تحلیل صدای تولید شده هنگام ضربه زدن به آن‌ها طراحی شده است. برای دستیابی به این هدف، ابتدا مجموعه‌ای از داده‌های صوتی شامل ضبط صداهای ضربه برای انواع مختلف مواد جمع‌آوری شده است. سپس از این داده‌ها برای آموزش یک مدل شبکه عصبی مصنوعی استفاده شده تا سیستم بتواند جنس مواد را به‌صورت خودکار و با دقت بالا پیش‌بینی کند.

اهمیت این پروژه نه‌تنها در کاربردهای صنعتی مانند شناسایی خودکار قطعات در خطوط تولید، بلکه در حوزه‌های دیگر مانند بازرسی کیفیت مواد، بازیافت هوشمند و حتی تشخیص مواد در محیط‌های غیرقابل دسترس مشهود است. استفاده از صدای تولید شده به عنوان یک داده کم‌هزینه و غیرتهاجمی برای شناسایی مواد، روش پیشنهادی را به راهکاری کارآمد و خلاقانه تبدیل می‌کند.

در این گزارش، فرآیند جمع‌آوری داده‌ها، طراحی و پیاده‌سازی مدل شبکه عصبی، و تحلیل نتایج به تفصیل بررسی شده است. هدف این گزارش، ارائه یک راهنمای جامع از مراحل انجام پروژه و بیان نقاط قوت و چالش‌های آن است.

معرفی پروژه

معرفی پروژه:

در این پروژه قصد داریم با استفاده از داده های صوتی ای که خودمان رکورد و ضبط کرده‌ایم، جنس موادی که به آن ضربه میزنیم را تشخیص دهیم. بدین منظور نیاز است که درمورد هر بخش به طور مجزا توضیحاتی ارائه دهیم:

مقایسه Classification و Regression کاربردها و تفاوت ها:

Classification و Regression دو وظیفه اصلی و متمایز در یادگیری ماشین هستند. هرچند هر دو به مدلسازی و پیش‌بینی مربوط می‌شوند، اما تفاوت‌های مهمی در هدف، خروجی، و کاربرد آن‌ها وجود دارد. به طور کلی زمانی که دیتاست محدود باشد از طبقه بندی انها استفاده میکنیم ولی اگه دیتاست خیلی بیشتر باشد و اصلا نمیتوانیم انها را طبقه بندی کنیم باید از رگرسیون استفاده کنیم.

Classification :

- هدف، پیش‌بینی کلاس یا دسته‌بندی داده است.
- داده‌های خروجی دسته‌ای یا گسسته (Discrete) هستند.
- مثال: تشخیص اینکه یک ایمیل "اسپم" است یا نه، یا پیش‌بینی جنس ماده مانند چوب یا فلز.

Regression :

- هدف، پیش‌بینی یک مقدار عددی (پیوسته) است.
- داده‌های خروجی پیوسته (Continuous) هستند.
- مثال: پیش‌بینی قیمت یک خانه یا دما در روز بعد.

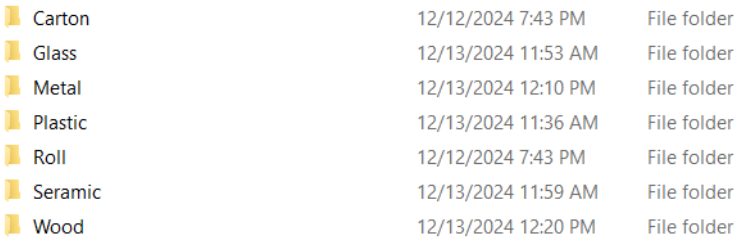
شکل۲- مقایسه Classification و Regression

دیتاست پروژه:

برای اجرای این پروژه و آموزش مدل هوش مصنوعی، نیاز به یک دیتاست مناسب از داده‌های صوتی داشتیم که بازتاب‌دهنده ویژگی‌های صوتی مرتبط با انواع مواد مختلف باشد. به همین منظور، مجموعه‌ای از داده‌ها شامل صدای ضربه زدن به مواد مختلف جمع‌آوری شده است. فرآیند جمع‌آوری داده‌ها به این صورت انجام شده است که با استفاده از یک ابزار فلزی یکسان، به هفت نوع ماده مختلف شامل موارد زیر ضربه زده‌ایم:

- چوب
- شیشه
- فلز
- سرامیک
- کارتون (مقوا)
- ورق فلزی نازک
- پلاستیک

برای هر ماده، چندین نمونه صوتی ضبط شده و در پوشه‌های جداگانه ذخیره شده است. این ساختار پوشه‌بندی، امکان سازماندهی بهتر داده‌ها و دسترسی آسان‌تر در مراحل پیش‌پردازش و آموزش مدل را فراهم می‌کند. شکل زیر ساختار پوشه‌ها و نحوه سازماندهی فایل‌های صوتی را نشان می‌دهد:



شکل۱- فولدر های دیتاست موجود در فایل ارائه پروژه

هر فایل صوتی در این دیتاست دارای کیفیت مناسب و با نرخ نمونه‌برداری ثابت ضبط شده است تا از یکنواختی داده‌ها اطمینان حاصل شود. همچنین، تلاش شده تا شرایط محیطی مانند نویز پس‌زمینه کنترل شود تا تأثیر عوامل مزاحم کاهش یابد.

این دیتاست به‌گونه‌ای طراحی شده است که شامل ویژگی‌های صوتی منحصربه‌فرد هر ماده باشد، که مدل بتواند با استفاده از آن‌ها جنس ماده را تشخیص دهد. در مجموع، این داده‌ها نقش اساسی در دقت و عملکرد مدل هوش مصنوعی ایفا خواهند کرد.

چالش‌هایی که در انجام این پروژه به آن برخوردیم:

اجرای پروژه‌ای که هدف آن تشخیص جنس مواد بر اساس صدای ضربه است، با چالش‌های مختلفی روبه‌روست. این چالش‌ها می‌توانند بر دقت و کارایی مدل تأثیر بگذارند. در ادامه، چالش‌های اصلی این پروژه توضیح داده شده است:

کیفیت و کمیت داده‌ها:

- نبود قطعات استاندارد
- تعداد کم داده‌ها
- تنوع ناکافی

نویز و عوامل محیطی

- نویز محیط
- انعکاس صدا
- تنوع تجهیزات ضبط

نیروی وارده به قطعات

- متغیر بودن شدت ضربه
- عدم یکنواختی

پیچیدگی جنس مواد

- شباهت‌های صوتی
- لایه‌بندی مواد

طراحی مدل یادگیری ماشین

- بیش‌پرازش (Overfitting)
- تطابق مدل با داده‌ها
- محاسبات سنگین

کد زده شده به زبان پایتون همراه توضیحات جزئی

کد زده شده به زبان پایتون همراه توضیحات جزئی:

سلول ۱:

```
import librosa
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import os
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten
from sklearn.utils.class_weight import compute_class_weight
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.utils.class_weight import compute_class_weight
```

در این بخش کتابخانه های مد نظر را اضافه کردیم...

توضیحات مربوط به کتابخانه librosa:

این کتابخانه برای پردازش سیگنال‌های صوتی استفاده می‌شود و ابزارهای متعددی برای تحلیل و استخراج ویژگی‌های صوتی فراهم می‌کند. برخی کاربردهای آن شامل:

- استخراج ویژگی‌هایی مثل MFCC، Spectrogram، و Chroma Features.
- بارگذاری فایل‌های صوتی و تغییر نرخ نمونه‌برداری (Resampling).
- تبدیل سیگنال‌های صوتی به نمایش‌های فرکانسی مثل STFT (تبدیل فوریه کوتاه مدت).

توضیحات مربوط به کتابخانه tensorflow:

یک کتابخانه یادگیری عمیق که برای ساخت و آموزش مدل‌های هوش مصنوعی استفاده می‌شود. ابزارهایی که در کد شما به چشم می‌خورد:

Sequential: برای ساخت مدل‌های لایه‌ای.

Dense: لایه کاملاً متصل برای شبکه‌های عصبی.

Conv1D و MaxPooling1D: برای پردازش داده‌های ترتیبی مثل صوت.

Flatten: برای صاف کردن خروجی‌های چندبعدی به ورودی لایه‌های Dense.

کاربرد tensorflow.keras.preprocessing.sequence.pad_sequences : این ابزار برای یکسان‌سازی طول داده‌های ترتیبی (مثل دنباله‌های ویژگی) با افزودن مقدارهای خالی (پدینگ) استفاده می‌شود.

توضیحات مربوط به کتابخانه sklearn:

این کتابخانه ابزارهای متنوعی برای پردازش داده، ارزیابی مدل، و انجام یادگیری ماشین ارائه می‌دهد. از موارد استفاده در کد شما:

train_test_split: برای تقسیم داده به مجموعه آموزش و آزمایش.

LabelEncoder: برای تبدیل برچسب‌های متنی به اعداد.

compute_class_weight: برای محاسبه وزن کلاس‌ها در دیتاست‌های نامتوازن.

confusion_matrix و classification_report: برای ارزیابی مدل با معیارهایی مثل دقت، بازخوانی و ماتریس سردرگمی.

توضیحات مربوط به کتابخانه Os:

این کتابخانه برای مدیریت فایل‌ها و مسیرها استفاده می‌شود. مثلاً:

- دسترسی به فایل‌های صوتی از پوشه‌های مشخص.
- پیمایش در دایرکتوری‌ها.

به طور خلاصه کتابخانه های فوق برای این موارد به کار رفته اند:

۱. پردازش داده‌های صوتی (استخراج ویژگی‌ها با librosa).
۲. ساخت مدل یادگیری عمیق (با TensorFlow).
۳. تقسیم داده‌ها و ارزیابی مدل (با sklearn).
۴. رفع مشکلات دیتاست‌های نامتوازن (با محاسبه وزن کلاس).
۵. نمایش نتایج و تجزیه‌وتحلیل مدل (با matplotlib و seaborn).

سلول ۲:

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

در این سلول، ما درایو گوگل خود را اضافه کرده ایم تا در ادامه فایل های مدنظر را از آن بخوانیم.

سلول ۳:

```
def preprocess_audio(audio, sr):
    audio = librosa.effects.preemphasis(audio)
    # audio = librosa.util.normalize(audio)
    return audio
```

این تابع بخشی از پیش‌پردازش صوتی است که معمولاً برای آماده‌سازی داده‌های صوتی برای مراحل بعدی، مانند استخراج ویژگی‌ها یا آموزش مدل، استفاده می‌شود. در آرگومان های این تابع، audio و sr وجود دارند که درمورد آن ها توضیح میدهیم:

- audio: سیگنال صوتی ورودی، به صورت یک آرایه عددی (NumPy array) که نمایانگر شدت سیگنال در نقاط زمانی مختلف است.
- sr: نرخ نمونه‌برداری (Sampling Rate) سیگنال صوتی که تعداد نمونه‌های صوتی در هر ثانیه را مشخص می‌کند.

این تابع از کتابخانه librosa برای اعمال پیش‌تأکید (Pre-emphasis) روی سیگنال صوتی استفاده می‌شود. پیش‌تأکید یک فیلتر فرکانسی ساده است که به تقویت فرکانس‌های بالا نسبت به فرکانس‌های پایین کمک می‌کند.

هدف این عملیات:

- **کاهش نویز:** فرکانس‌های پایین در صدا معمولاً دارای نویز بیشتری هستند.
- **افزایش وضوح ویژگی‌ها:** برخی ویژگی‌های صوتی مهم (مثل فرمنت‌ها) در فرکانس‌های بالاتر قرار دارند.
- **بهبود عملکرد مدل:** در مسائل یادگیری ماشین مرتبط با صوت، پیش‌تأکید می‌تواند داده‌های بهتری برای استخراج ویژگی‌ها فراهم کند.

`# audio = librosa.util.normalize(audio)` : این بخش کامنت شده است، اما اگر فعال باشد، از تابع normalize برای نرمال‌سازی سیگنال صوتی استفاده می‌شود.

هدف نرمال‌سازی:

- مقیاس سیگنال صوتی را به یک بازه مشخص (معمولاً بین -۱ و ۱) تغییر می‌دهد.
- از مقادیر بسیار بزرگ یا بسیار کوچک جلوگیری می‌کند که ممکن است در یادگیری مدل اختلال ایجاد کند.

سلول ٤:

```
# و سایر ویژگی‌ها از یک فایل صوتی MFCC استخراج ویژگی‌های
def extract_features(file_path):
    # بارگذاری فایل صوتی
    audio, sr = librosa.load(file_path, sr=None)

    # استخراج ویژگی‌های MFCC
    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
    mfcc_mean = np.mean(mfcc, axis=1)

    # استخراج ویژگی‌های Spectral Contrast
    spectral_contrast = librosa.feature.spectral_contrast(y=audio, sr=sr)
    spectral_contrast_mean = np.mean(spectral_contrast, axis=1)

    # استخراج ویژگی‌های Chroma
    chroma = librosa.feature.chroma_stft(y=audio, sr=sr)
    chroma_mean = np.mean(chroma, axis=1)


    #Zero Crossing Rate (ZCR): تعداد عبورهای صفر سیگنال.
    mel_spec = librosa.feature.melspectrogram(y=audio, sr=sr)
    mel_spec_mean = np.mean(mel_spec, axis=1)


    # ویژگی‌های Zero Crossing Rate
    zcr = librosa.feature.zero_crossing_rate(audio)
    zcr_mean = np.mean(zcr)

  # ویژگی‌های فلاتر
    flatness = librosa.feature.spectral_flatness(y=audio)
    flatness_mean = np.mean(flatness)

    # # ویژگی‌های انرژی
    # energy = np.sum(audio**2) / len(audio)

    # # ویژگی‌های رول‌آف فرکانسی
```

```
    # spectral_rolloff = librosa.feature.spectral_rolloff(y=audio, sr=sr,
roll_percent=0.85)
    # spectral_rolloff_mean = np.mean(spectral_rolloff)

    # # ویژگی‌های عرض باند
    # bandwidth = librosa.feature.spectral_bandwidth(y=audio, sr=sr)
    # bandwidth_mean = np.mean(bandwidth)


    # #استخراج فرکانس غالب (Dominant Frequency)
    # fft = np.fft.fft(audio)
    # freqs = np.fft.fftfreq(len(fft), 1/sr)
    # magnitude = np.abs(fft)
    # dominant_freq = freqs[np.argmax(magnitude[:len(magnitude)//2])]  #
فرکانس غالب

    # # استخراج میانگین فرکانس (Mean Frequency)
    # weighted_sum = np.sum(freqs[:len(magnitude)//2] *
magnitude[:len(magnitude)//2])
    # mean_freq = weighted_sum / np.sum(magnitude[:len(magnitude)//2])


    #-------------------------------------------------------------------
-----------------------------------------------------------

    features = np.hstack([
         mfcc_mean,
         spectral_contrast_mean,      # کنتراست طیفی
         chroma_mean ,
         mel_spec_mean  ,              #تعداد عبورهای صفر سیگنال
         zcr_mean   ,                  # کروماتیک
         flatness_mean,                # فلاتر طیفی

        #energy,                       # انرژی
        # spectral_rolloff_mean,       # رول‌آف فرکانسی
        # bandwidth_mean,              # عرض باند
        # dominant_freq,               # فرکانس غالب
        #mean_freq                     # میانگین فرکانس
    ])

    return features
```

MFCC یکی از مهم‌ترین ویژگی‌های صوتی است که الگوهای طیفی در مقیاس مل را مدل می‌کند. n_mfcc=13 نشان می‌دهد که ۱۳ ضرایب MFCC محاسبه می‌شود. میانگین این ضرایب محاسبه شده و به‌عنوان ویژگی نهایی ذخیره می‌شود.

spectral_contrast : تفاوت شدت فرکانس‌ها در باندهای مختلف را اندازه‌گیری می‌کند و به تشخیص تفاوت‌های بین مناطق فرکانسی کمک می‌کند.

کروماتیک توزیع انرژی در ۱۲ گام موسیقی را نمایش می‌دهد. در اینجا استفاده از axis=1 مهم است زیرا:

- ✓ میانگین انرژی برای هر گام کروماتیک (به‌صورت جداگانه) در طول کل بازه زمانی محاسبه می‌شود.
- ✓ خروجی یک بردار ۱۲-بعدی است که اطلاعات خاص هر گام را حفظ می‌کند.

Zero Crossing Rate (ZCR): تعداد عبورهای صفر سیگنال (از مثبت به منفی یا برعکس) را اندازه‌گیری می‌کند و برای تمایز بین سیگنال‌های موسیقی و صداهای کوبه‌ای مفید است.

Spectral Flatness (فلاتر طیفی): یکنواختی طیف را اندازه‌گیری می‌کند. مقدار بالای فلاتر نشان‌دهنده نویز یا صداهای یکنواخت است.

Features : ویژگی‌های استخراج‌شده در قالب یک بردار با استفاده از np.hstack ترکیب می‌شوند

برخی ویژگی‌ها در این سلول از کد کامنت شده‌اند:

- Spectral Rolloff: نقطه‌ای که در آن انرژی طیفی به درصد خاصی (مثلاً ۸۵٪) کاهش می‌یابد.
- Spectral Bandwidth: میزان گستردگی فرکانس‌ها.
- Energy: انرژی کل سیگنال.
- Dominant Frequency: فرکانس غالب در سیگنال.
- Mean Frequency: میانگین فرکانس‌ها.

سلول ۵:

```
# def augment_time_stretch(audio, sr, rates=[0.8, 1.2]):
#     augmented_audios = []
#     for rate in rates:
#         augmented_audios.append(librosa.effects.time_stretch(audio,
rate))
#     return augmented_audios
```

ما در این بخش از کد تابعی تعریف کرده ایم که اگر دیتاست و داده ها کوتاه و کم بودند، کشش زمانی داشته باشیم. این تابع، سرعت صوت را به ۰.۸ و ۱.۲ تغییر میداد و آن را سیو میکرد تا داده ها را زیاد کند.

سلول ٦:

```
def load_data(audio_folder):
    features = []
    labels = []
    filenames = []

    # خواندن تمام فایل‌ها در پوشه
    for label in os.listdir(audio_folder):
        label_folder = os.path.join(audio_folder, label)
        if os.path.isdir(label_folder):
            for file in os.listdir(label_folder):
                if file.endswith(".wav"):
                    file_path = os.path.join(label_folder, file)
                    feature = extract_features(file_path)
                    features.append(feature)
                    labels.append(label)
                    filenames.append(file)


    return np.array(features), np.array(labels), filenames
```

این سلول وظیفه بارگذاری داده‌های صوتی از پوشه درون گوگل درایو، استخراج ویژگی‌ها از این داده‌ها، و لیبل زنی داده ها را بر عهده دارد. لیست ویژگی‌ها به یک آرایه NumPy تبدیل شده و همراه با لیبل ها و نام فایل‌ها برگردانده می‌شود.

خروجی:

- features: آرایه‌ای شامل ویژگی‌های استخراج‌شده از فایل‌های صوتی.
- labels: آرایه‌ای از برچسب‌های متنی (نام دسته‌بندی).
- filenames: لیستی از نام فایل‌های صوتی.

این سلول فرض می‌کند که پوشه صوتی (مثلاً audio_folder) ساختاری شبیه به زیر دارد:

شکل۳- مسیر فایل ها و دایرکتوری داده‌ها (دیتاست)

سلول ۷:

```
audio_folder = r"/content/drive/MyDrive/Colab
Notebooks/AI_Project_AudioDetection/finally_data"
X, y , filenames = load_data(audio_folder)
```

بارگذاری داده‌ها از پوشه داده‌های صوتی در این سلول انجام میشود. نکته ای که حائز اهمیت و یادآوری است این است که ما از Raw String ها باید در اینجا استفاده کنیم که پایتون \ ها را به عنوان دستورات نشناسد.

سلول ۸:

```
encoder = LabelEncoder()
y_encoded = encoder.fit_transform(y)
```

این کد با استفاده از کلاس LabelEncoder از کتابخانه sklearn، برچسب‌های متنی را به مقادیر عددی تبدیل می‌کند. این فرآیند رمزگذاری برچسب‌ها (Label Encoding) نامیده می‌شود و برای آماده‌سازی برچسب‌ها برای مدل‌های یادگیری ماشین مورد استفاده قرار می‌گیرد. لیبل انکودر، یک راه سریع و ساده برای تبدیل مقادیر متنی به عددی است.

چرا از این روش استفاده می‌شود؟

مدل‌های یادگیری ماشین (مانند شبکه‌های عصبی یا SVM) با داده‌های عددی کار می‌کنند و نمی‌توانند مقادیر متنی را مستقیماً پردازش کنند.

سلول ۹:

```
X_train, X_test, y_train, y_test, filenames_train, filenames_test =
train_test_split(X, y_encoded, filenames, test_size=0.2, random_state=41

          )

#شبکه عصبی پیچیده
model = Sequential()
model.add(Conv1D(32, kernel_size=3, activation='relu',
input_shape=(X_train.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(len(np.unique(y_encoded)), activation='softmax'))


model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# آموزش مدل
class_weights = compute_class_weight('balanced',
classes=np.unique(y_train), y=y_train)
class_weights_dict = dict(enumerate(class_weights))

# آموزش مدل و ذخیره تاریخچه آموزش
history = model.fit(X_train, y_train, epochs=60, batch_size=32,
validation_data=(X_test, y_test), class_weight=class_weights_dict)#,
callbacks=[early_stopping])

# ارزیابی مدل
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

در این سلول، شبکه عصبی کانولوشنی خود را مینویسیم. ابتدا داده ها را به داده های تست و آموزش تقسیم کرده و سپس آن را آموزش میدهیم.

random_state=41: تعیین یک دانه (seed) ثابت برای تقسیم دادهها به طور تصادفی. این باعث می‌شود که نتایج هر بار ثابت باشد.

Sequential: به این معناست که لایه‌ها به ترتیب و به صورت خطی اضافه خواهند شد. در این نوع مدل‌ها، لایه‌ها به صورت تسلسلی به هم متصل هستند.

افزودن لایه‌ها به مدل:

لایه اول: Conv1D (لایه کانولوشن یک‌بعدی)

- ۳۲: تعداد فیلترهای کانولوشن. هر فیلتر یک ویژگی خاص از داده‌ها را استخراج می‌کند. در اینجا ۳۲ فیلتر برای استخراج ویژگی‌های مختلف از سیگنال صوتی استفاده می‌شود.
- kernel_size=3: اندازه کرنل (یا فیلتر). این مشخص می‌کند که هر فیلتر به چند عنصر از ورودی نگاه خواهد کرد. در اینجا کرنل ۳ (سه عنصر) است.
- input_shape=(X_train.shape[1], 1): اندازه ورودی به لایه کانولوشنی. چون ورودی شما یک آرایه دوبعدی است، X_train.shape[۱] تعداد ویژگی‌ها (ویژگی‌های صوتی) را نشان می‌دهد و ۱ به این معنا است که ورودی‌ها یک‌بعدی هستند (تک‌کانال).

لایه دوم: MaxPooling1D (لایه ماکس‌پولینگ یک‌بعدی)

- MaxPooling1D(pool_size=2): این لایه برای کاهش ابعاد داده‌ها و استخراج ویژگی‌های برجسته‌تر استفاده می‌شود. در اینجا از ماکس‌پولینگ با اندازه ۲ استفاده شده است، به این معنا که از میانگین بزرگ‌ترین مقدار در هر دو عنصر پیاپی انتخاب می‌شود. این باعث کاهش ابعاد و افزایش ویژگی‌های مهم می‌شود.

لایه سوم: Flatten (لایه صاف‌سازی)

- Flatten(): این لایه داده‌های چندبعدی را به یک آرایه یک‌بعدی تبدیل می‌کند تا بتوان آن‌ها را به لایه‌های Dense وارد کرد.

لایه چهارم: Dense (لایه چگال)

- Dense: این یک لایه کاملاً متصل است که در آن هر نورون به همه نورون‌های لایه قبلی متصل است.
- ۳۲: تعداد نورون‌های لایه. این لایه دارای ۳۲ نورون است که اطلاعات استخراج‌شده از لایه‌های قبلی را پردازش می‌کنند.

لایه پنجم: Dense (لایه چگال نهایی)

- len(np.unique(y_encoded)): تعداد نورون‌ها معادل تعداد کلاس‌ها (برچسب‌های یکتا) است. برای هر برچسب (کلاس) یک نورون در این لایه وجود دارد.

- activation='softmax': تابع فعال‌سازی Softmax برای تبدیل خروجی‌ها به احتمال‌های نرمال‌شده بین ۰ و ۱ استفاده می‌شود. این تابع برای مسائل طبقه‌بندی چندکلاسه مناسب است.

کامپایل مدل:

- optimizer='adam': از الگوریتم Adam برای بهینه‌سازی استفاده می‌شود. این الگوریتم برای یادگیری سریع‌تر و پایدارتر معروف است.
- loss='sparse_categorical_crossentropy': از Cross-Entropy برای محاسبه میزان خطا در مدل استفاده می‌شود. چون y_encoded به صورت عددی است، از sparse_categorical_crossentropy استفاده می‌شود.

آموزش مدل:

- compute_class_weight('balanced', ...): برای مقابله با مشکلات عدم توازن کلاس‌ها (اگر تعداد نمونه‌های برخی کلاس‌ها بسیار کم یا زیاد باشد)، از وزن‌های متعادل استفاده می‌شود. این کار باعث می‌شود که مدل برچسب‌های کم‌نمونه را بیشتر در نظر بگیرد.
- epochs=60: تعداد دفعاتی که مدل باید روی داده‌ها آموزش ببیند.
- batch_size=32: اندازه دسته‌ای که در هر مرحله از آموزش به مدل داده می‌شود.
- validation_data=(X_test, y_test): داده‌های تست برای ارزیابی مدل در طول فرآیند آموزش استفاده می‌شود.
- class_weight=class_weights_dict: وزن‌های متعادل برای کلاس‌ها برای مقابله با مشکلات توازن کلاس‌ها به مدل داده می‌شود.

و در انتها، ارزیابی مدل:

- ✓ evaluate(X_test, y_test): مدل را روی داده‌های تست ارزیابی می‌کند و دو مقدار loss و accuracy را برمی‌گرداند.

هنگام اجرای این سلول، خروجی زیر را دریافت خواهیم کرد که بعضا بستگی به شانس، درصد دقت آن، ۱۰۰ یا ۹۶.۶۹ گزارش میشود. این نشان میدهد که کد به خوبی بر روی این دیتاست اندک ما آموزش دیده و آماده پاسخگویی و تست نتایج جدید خواهد بود. خروجی اجرای این سلول نیز در صفحه بعدی آورده شده است:

```
Epoch 1/60
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
4/4 ━━━━━━━━━━━━━━━━━━━━ 2s 70ms/step - accuracy: 0.1832 - loss: 6.4155 - val_accuracy: 0.2667 - val_loss: 3.2810
Epoch 2/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.3431 - loss: 3.3911 - val_accuracy: 0.4000 - val_loss: 1.7414
Epoch 3/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.3124 - loss: 2.2778 - val_accuracy: 0.5667 - val_loss: 1.2060
Epoch 4/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.5021 - loss: 1.3118 - val_accuracy: 0.5333 - val_loss: 1.0818
Epoch 5/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.4951 - loss: 1.2775 - val_accuracy: 0.6000 - val_loss: 0.9745
Epoch 6/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.5591 - loss: 1.0044 - val_accuracy: 0.7333 - val_loss: 0.7754
Epoch 7/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6600 - loss: 0.7897 - val_accuracy: 0.7667 - val_loss: 0.6797
Epoch 8/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.7405 - loss: 0.7517 - val_accuracy: 0.7000 - val_loss: 0.8511
Epoch 9/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.7335 - loss: 0.7305 - val_accuracy: 0.8333 - val_loss: 0.4825
Epoch 10/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.7371 - loss: 0.6328 - val_accuracy: 0.8667 - val_loss: 0.4356
Epoch 11/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8030 - loss: 0.5229 - val_accuracy: 0.7667 - val_loss: 0.5975
Epoch 12/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8497 - loss: 0.4672 - val_accuracy: 0.9000 - val_loss: 0.3755
Epoch 13/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8434 - loss: 0.4473 - val_accuracy: 0.9000 - val_loss: 0.3435
Epoch 14/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.8979 - loss: 0.3501 - val_accuracy: 0.9000 - val_loss: 0.3240
Epoch 15/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9140 - loss: 0.2989 - val_accuracy: 0.9333 - val_loss: 0.2915
Epoch 16/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9466 - loss: 0.2689 - val_accuracy: 0.9333 - val_loss: 0.2917
Epoch 17/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9263 - loss: 0.2794 - val_accuracy: 0.9667 - val_loss: 0.2460
Epoch 18/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9630 - loss: 0.2469 - val_accuracy: 0.9333 - val_loss: 0.2553
Epoch 19/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9302 - loss: 0.2424 - val_accuracy: 0.9333 - val_loss: 0.2437
Epoch 20/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9435 - loss: 0.2313 - val_accuracy: 0.9333 - val_loss: 0.2201
Epoch 21/60
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9471 - loss: 0.2141 - val_accuracy: 0.9333 - val_loss:
0.2348
Epoch 22/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9067 - loss: 0.2197 - val_accuracy: 0.9667 - val_loss:
0.2044
Epoch 23/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9818 - loss: 0.2043 - val_accuracy: 0.9333 - val_loss:
0.2487
Epoch 24/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9476 - loss: 0.2058 - val_accuracy: 1.0000 - val_loss:
0.1759
Epoch 25/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9805 - loss: 0.1837 - val_accuracy: 0.9333 - val_loss:
0.2005
Epoch 26/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9599 - loss: 0.1802 - val_accuracy: 0.9333 - val_loss:
0.1921
Epoch 27/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9427 - loss: 0.1660 - val_accuracy: 1.0000 - val_loss:
0.1597
Epoch 28/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9791 - loss: 0.1353 - val_accuracy: 0.9333 - val_loss:
0.2144
Epoch 29/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9583 - loss: 0.1527 - val_accuracy: 0.9667 - val_loss:
0.1466
Epoch 30/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9732 - loss: 0.1416 - val_accuracy: 0.9333 - val_loss:
0.1808
Epoch 31/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9742 - loss: 0.1321 - val_accuracy: 1.0000 - val_loss:
0.1461
Epoch 32/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9966 - loss: 0.1228 - val_accuracy: 0.9667 - val_loss:
0.1472
Epoch 33/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9914 - loss: 0.1205 - val_accuracy: 1.0000 - val_loss:
0.1398
Epoch 34/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9914 - loss: 0.1159 - val_accuracy: 0.9667 - val_loss:
0.1419
Epoch 35/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9750 - loss: 0.1061 - val_accuracy: 0.9667 - val_loss:
0.1346
Epoch 36/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.1018 - val_accuracy: 1.0000 - val_loss:
0.1273
Epoch 37/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.1111 - val_accuracy: 0.9333 - val_loss:
0.1554
Epoch 38/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9575 - loss: 0.1031 - val_accuracy: 1.0000 - val_loss:
0.1184
Epoch 39/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9685 - loss: 0.1210 - val_accuracy: 1.0000 - val_loss:
0.1063
Epoch 40/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9617 - loss: 0.1036 - val_accuracy: 0.9333 - val_loss:
0.2034
Epoch 41/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9390 - loss: 0.1116 - val_accuracy: 1.0000 - val_loss:
0.1018
Epoch 42/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9766 - loss: 0.1175 - val_accuracy: 1.0000 - val_loss:
0.1077
```

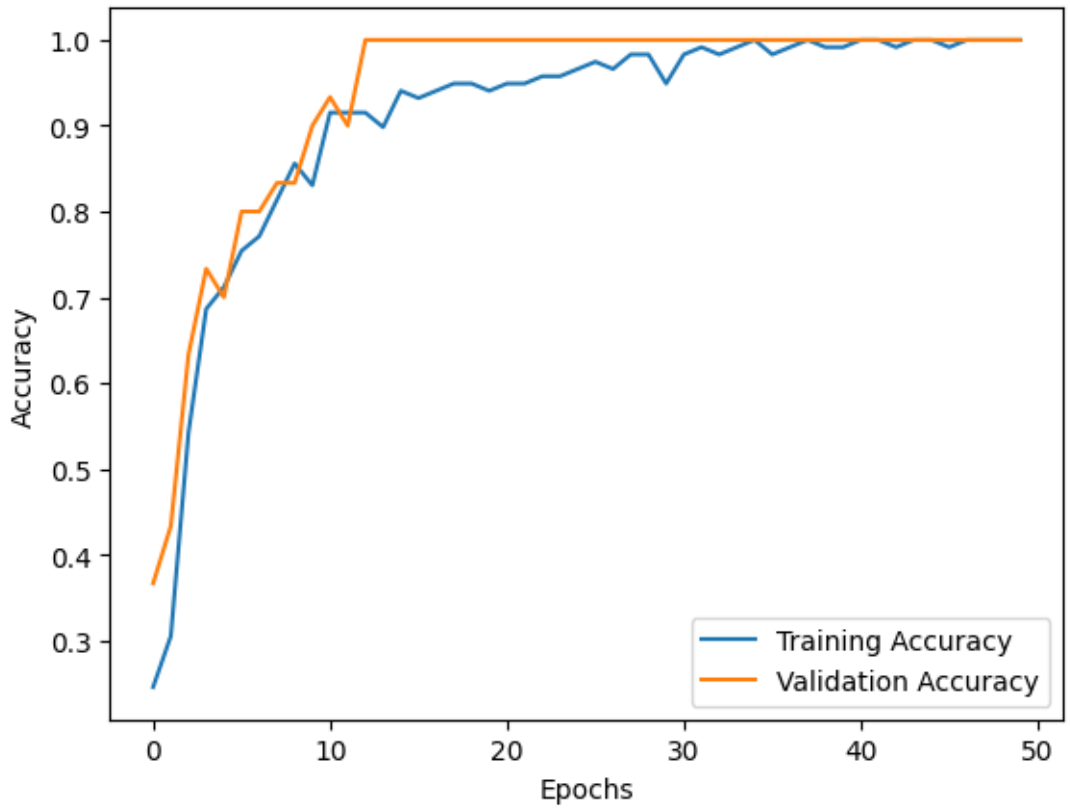
```
Epoch 43/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0762 - val_accuracy: 0.9667 - val_loss:
0.1218
Epoch 44/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9742 - loss: 0.0828 - val_accuracy: 1.0000 - val_loss:
0.1097
Epoch 45/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 1.0000 - loss: 0.0754 - val_accuracy: 1.0000 - val_loss:
0.1033
Epoch 46/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.0670 - val_accuracy: 1.0000 - val_loss:
0.1071
Epoch 47/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.0708 - val_accuracy: 1.0000 - val_loss:
0.0992
Epoch 48/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.0620 - val_accuracy: 1.0000 - val_loss:
0.0967
Epoch 49/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.0658 - val_accuracy: 1.0000 - val_loss:
0.0935
Epoch 50/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0730 - val_accuracy: 1.0000 - val_loss:
0.0969
Epoch 51/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.0646 - val_accuracy: 1.0000 - val_loss:
0.0909
Epoch 52/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.0546 - val_accuracy: 1.0000 - val_loss:
0.0895
Epoch 53/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0641 - val_accuracy: 1.0000 - val_loss:
0.1005
Epoch 54/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9914 - loss: 0.0624 - val_accuracy: 1.0000 - val_loss:
0.0863
Epoch 55/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.0634 - val_accuracy: 1.0000 - val_loss:
0.0741
Epoch 56/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0601 - val_accuracy: 0.9667 - val_loss:
0.0983
Epoch 57/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.9852 - loss: 0.0628 - val_accuracy: 1.0000 - val_loss:
0.0935
Epoch 58/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0565 - val_accuracy: 1.0000 - val_loss:
0.0725
Epoch 59/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 1.0000 - loss: 0.0577 - val_accuracy: 0.9667 - val_loss:
0.0934
Epoch 60/60
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 1.0000 - loss: 0.0577 - val_accuracy: 1.0000 - val_loss:
0.0691
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 1.0000 - loss: 0.0691
Test Accuracy: 100.00%
```

خروجی سلول شماره ۱۰ و آموزش مدل شبکه عصبی کانولوشنی

سلول ۱۰:

```
# رسم نمودار دقت بر حسب epochها
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

این سلول، نمودار زیر را به ما خروجی خواهد داد که نشان‌دهنده دقت مدل در داده‌های آموزشی و اعتبارسنجی در طول دوره‌های مختلف آموزش است که همانطور که میبینید در انتها به یک همگرا شده اند.



شکل۴- خروجی نمودار تغییرات دقت برحسب هر اپوک

سلول ۱۱:

```
class_names = ['carton', 'Glass', 'metal', 'Pelastic', 'Roll', 'Seramic',
'Wood']  # نام کلاس‌ها

#  نمایش یکی از نمونه‌های نام فایل برای هر کلاس برای پیدا کردن نام دسته
for label in np.unique(y_test):
    sample_idx = np.where(y_test == label)[0][0]  # گرفتن ایندکس اولین
نمونه
    print(f"Class {class_names[label]}: Example filename ->
{filenames_test[sample_idx]}")

# پیش‌بینی دسته‌ها
y_pred = np.argmax(model.predict(X_test), axis=1)

# نمایش ماتریس
cm = confusion_matrix(y_test, y_pred)


#گزارش عملکرد
print(classification_report(y_test, y_pred, target_names=class_names))
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

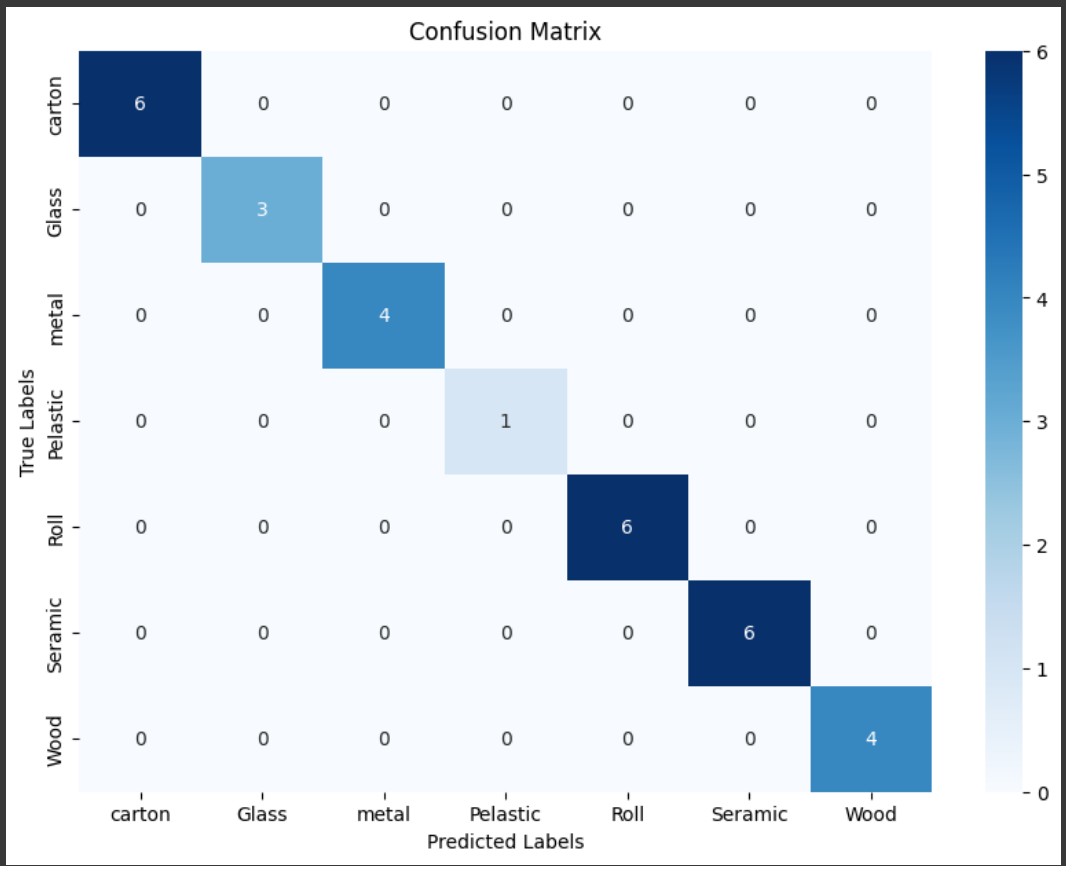
در این سلول، نمایش نمونه‌های پیش‌بینی‌شده، نمایش confusion matrix، و گزارش عملکرد (classification report) انجام شده.

نتیجه و خروجی این سلول (سلول ۱۱) در زیر قابل مشاهده است:

```
Class carton: Example filename -> Carton_3 - Copy.wav
Class Glass: Example filename -> Glass_55.wav
Class metal: Example filename -> Metal_68.wav
Class Pelastic: Example filename -> Plastic_55.wav
Class Roll: Example filename -> Roll_9.wav
Class Seramic: Example filename -> Seramic_09.wav
Class Wood: Example filename -> Wood_4.wav
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 93ms/step
              precision    recall  f1-score   support

      carton       1.00      1.00      1.00         6
       Glass       1.00      1.00      1.00         3
       metal       1.00      1.00      1.00         4
    Pelastic       1.00      1.00      1.00         1
        Roll       1.00      1.00      1.00         6
     Seramic       1.00      1.00      1.00         6
        Wood       1.00      1.00      1.00         4

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```



شکل۵- خروجی ماتریکس کانفیوژن برای دیدن پیش بینی ها

سلول ۱۲:

```
# پیش‌بینی برچسب جنسیت
predictions = model.predict(X_test)

# تبدیل پیش‌بینی‌ها به برچسب‌های اصلی
predicted_labels = encoder.inverse_transform(np.argmax(predictions,
axis=1))
true_labels = encoder.inverse_transform(y_test)


#نمایش تست
num_samples = max(10, len(X_test))
for i in range(num_samples):
    print(f"Sample {i+1}:")
    print(f"File: {filenames_test[i]}")
    print(f"True label: {true_labels[i]}")
    print(f"Predicted label: {predicted_labels[i]}")
    print("-" * 50)
```

این سلول برای پیش‌بینی و ارزیابی عملکرد مدل در پیش‌بینی برچسب‌های کلاس‌ها (مثلاً جنسیت یا هر نوع برچسب دیگر) استفاده می‌شود.

np.argmax(predictions, axis=1): این دستور برای پیدا کردن کلاس با بالاترین احتمال برای هر نمونه در پیش‌بینی‌ها استفاده می‌شود. در واقع، با توجه به احتمالات پیش‌بینی شده، این دستور کلاسی را که احتمال بیشتری دارد انتخاب می‌کند.

encoder.inverse_transform(...): این تابع از LabelEncoder برای تبدیل مقادیر عددی (که از مدل خروجی گرفته شده) به برچسب‌های اصلی (متنی) استفاده می‌کند. به عبارت دیگر، پیش‌بینی‌های عددی به نام کلاس‌های واقعی تبدیل می‌شوند.

سلول ۱۳:

```
def predict_sound_type_with_reasons(file_path):
    # بارگذاری فایل صوتی
    audio, sr = librosa.load(file_path, sr=None)

    # اعمال حذف نویز
    audio = preprocess_audio(audio, sr)

    # استخراج ویژگی‌ها از فایل صوتی جدید
    features = extract_features(file_path)

    # تغییر شکل ویژگی‌ها به صورت مناسب برای مدل
    features = features.reshape(1, -1)

    # پیش‌بینی با مدل
    predictions = model.predict(features)

    # سه پیش‌بینی برتر و درصدهایشان
    top_3_indices = np.argsort(predictions[0])[::-1][:3]  # ایندکس‌های سه پیش‌بینی برتر
    top_3_probabilities = predictions[0][top_3_indices]  # درصد شباهت سه پیش‌بینی برتر

    # تبدیل ایندکس‌ها به نام جنس
    top_3_names = encoder.inverse_transform(top_3_indices)

    # دلایل انتخاب (بر اساس ویژگی‌های برجسته)
    dominant_features = features[0]  # ویژگی‌های ورودی
    important_features = np.argsort(dominant_features)[::-1][:7]  # سه ویژگی مهم

    reasons = [
        f"Feature {i} with value {dominant_features[i]:.4f}"
        for i in important_features
    ]

    return {
        "Top Predictions": [
            {"Type": name, "Similarity": f"{prob*100:.2f}%", "Reason": reasons}
            for name, prob in zip(top_3_names, top_3_probabilities)
        ]
    }
```

این سلول (سلول ۱۳) برای پیش‌بینی نوع صدا از یک فایل صوتی خاص و ارائه دلایل انتخاب بر اساس ویژگی‌های برجسته آن طراحی شده است.

sr=None به این معنی است که نمونه‌برداری (sampling rate) فایل صوتی به طور پیش‌فرض نگه‌داشته می‌شود.

features.reshape(1, -1) : در اینجا کاری میکند که یک نمونه فایل صوتی در یک درایه قرار گیرد.

این تابع برای یک فایل صوتی، سه پیش‌بینی برتر از مدل را می‌دهد و دلایل انتخاب آن‌ها را بر اساس ویژگی‌های برجسته ورودی توضیح می‌دهد.

سلول ۱۴:

```
# تست مدل با فایل صوتی جدید
test_file = r"/content/drive/MyDrive/Colab
Notebooks/AI_Project_AudioDetection/test/قوری در.wav"  # مسیر به فایل
صوتی تست شما
result = predict_sound_type_with_reasons(test_file)

# نمایش سه پیشبینی برتر
for idx, prediction in enumerate(result["Top Predictions"], start=1):
    print(f"Prediction {idx}:")
    print(f"  Type: {prediction['Type']}")
    print(f"  Similarity: {prediction['Similarity']}")
    print()

print(f"  Reasons: {', '.join(prediction['Reason'])}")
```

کد پیش‌بینی نوع صدا (کلاس) برای یک فایل صوتی جدید انجام می‌دهد و سه پیش‌بینی برتر را به همراه درصد شباهت و دلایل انتخاب آن‌ها به صورت قابل فهم نمایش می‌دهد. این به کاربر کمک می‌کند تا علاوه بر نتیجه پیش‌بینی، دلیل انتخاب آن را نیز بفهمد.

در بخش بعدی، تست کد و خروجی داده های تست را میتوانید مشاهده کنید....

تست کد و پیش بینی صدای ضربه به قطعات جدید

تست کد و پیش بینی صدای ضربه به قطعات جدید:

تست اول:

با استفاده از سلول آخر (سلول شماره ۱۴) و داد آدرس زیر، به خروجی زیر دست یافتیم:

```
test_file = r"/content/drive/MyDrive/Colab
Notebooks/AI_Project_AudioDetection/test/در قوری.wav"
```

نتیجه:

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step
Prediction 1:
  Type: Metal
  Similarity: 95.68%
Prediction 2:
  Type: Pelastic
  Similarity: 1.96%
Prediction 3:
  Type: Carton
  Similarity: 1.43%
  Reasons: Feature 1 with value 84.2023, Feature 3 with value 36.4675,
Feature 2 with value 19.4242, Feature 13 with value 19.2450, Feature 18
with value 17.7276, Feature 19 with value 17.1920, Feature 17 with value
15.9307
```

که نشان میدهد توانستیم صدای در قوری فلزی را تشخیص دهیم.

تست دوم:

```
test_file = r"/content/drive/MyDrive/Colab
Notebooks/AI_Project_AudioDetection/test/پایه میز.wav"
```

نتیجه:

```
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
Prediction 1:
  Type: Metal
  Similarity: 99.45%
Prediction 2:
  Type: Pelastic
  Similarity: 0.44%
Prediction 3:
  Type: Carton
  Similarity: 0.05%
  Reasons: Feature 1 with value 72.5025, Feature 3 with value
30.5587, Feature 13 with value 18.0171, Feature 18 with value 16.8498,
Feature 19 with value 16.4302, Feature 17 with value 15.9818, Feature 16
with value 14.4989
```

که مجددا نشان میدهد در تست دوم نیز مدل ما به خوبی توانسته جنس ماده را علی رغم کمبود دیتا برای آموزش، تشخیص دهد.

در ادامه تست های دیگری نیز انجام شدند که مدل به خوبی نیز آن ها را تشخیص داد که به جهت دوری از کثرت در نوشتار، از آوردن آنها در این گزارش خودداری شد اما در ارائه درصورت وجود زمان به آن ها نیز خواهیم پرداخت.

منابع:

- ✓ دیتاست این پروژه در کارگاه‌ها و مکان های زیر تهیه شده است:
  - کارگاه ماشین ابزار دانشگاه علم و صنعت ایران
  - کارگاه اتومکانیک دانشگاه علم و صنعت ایران
  - کلاس درس هوش مصنوعی (کلاس ۱۱۶)
  - دانشکده مهندسی معماری دانشگاه علم و صنعت
  - خوابگاه دانشجویی داخل
- ✓ استفاده از chatgpt برای نوشتن برخی سلول‌ها
- ✓ Deep Learning for Audio Classification