Summery of the code

Here's a summarized explanation of the code, broken down into sections:

---

1. Load and Preprocess Audio Data

Purpose:

- Load audio files from the dataset.
- Extract meaningful features for model input.

Code Breakdown:

1. **Iterate Through Dataset**:
    - Loops through folders `Animal` and `Environment` to label each category (`0` for Animal, `1` for Environment).
2. **Load Audio Files**:
    - Uses `librosa` to load `.wav` audio files.
3. **Feature Extraction**:
    - Computes the **Mel spectrogram** for each audio file.
    - Reduces dimensionality by taking the **mean** of each feature across time.
4. **Return Data**:
    - Returns `X` (features) and `y` (labels) as NumPy arrays.

Output:

- `X`: Feature matrix of shape `(number_of_samples, number_of_features)`.
- `y`: Labels array of shape `(number_of_samples,)`.

---

2. Build a 1D CNN Model

Purpose:

- Define a 1D Convolutional Neural Network for audio classification.

Code Breakdown:

1. **Model Architecture**:
    - **Conv1D Layer**: Extracts patterns from 1D feature input.
    - **MaxPooling1D**: Reduces the feature map size to prevent overfitting.
    - **Flatten**: Converts 2D features into 1D for Dense layers.
    - **Dense Layers**:

        - First layer learns hidden patterns with 64 neurons.
        - Final layer outputs probabilities for 2 categories (`Animal`, `Environment`).
2. **Compilation**:
    - Optimizer: **Adam** for efficient training.
    - Loss: **Sparse Categorical Cross-Entropy** for labeled classification.
    - Metric: Accuracy.

---

3. Perform Cross-Validation

Purpose:

- Evaluate the model's consistency across different dataset splits using k-fold cross-validation.

Code Breakdown:

1. **KFold Split**:
    - Splits the dataset into `k` folds (default: 5).
    - Uses `k-1` folds for training and 1 fold for validation in each iteration.
2. **Train and Evaluate**:
    - Trains the model on the training folds.
    - Evaluates accuracy on the validation fold.
3. **Collect Metrics**:
    - Computes **mean accuracy** and **standard deviation** across folds.

Output:

- `mean_acc`: Average accuracy across folds.
- `std_dev`: Variability in performance across folds.

---

4. Perform Validation Split

Purpose:

- Evaluate the model's performance using a single train-test split.

Code Breakdown:

1. **Train-Test Split**:
    - Splits the dataset into 80% training and 20% validation.
2. **Train and Evaluate**:
    - Trains the model on the training set.
    - Evaluates accuracy on the validation set.

Output:

- `val_acc`: Accuracy on the validation set.

---

5. Visualize Results

Purpose:

- Compare Cross-Validation and Validation Split results visually.

Code Breakdown:

1. **Bar Chart**:
   - Plots accuracy for Cross-Validation (mean) and Validation Split.
   - Colors:
     - Blue: Cross-Validation.
     - Green: Validation Split.
2. **Title and Labels**:
   - Adds appropriate labels for comparison.

Final Results

- **Cross-Validation**: Provides robust evaluation with mean accuracy and variability (std dev).
- **Validation Split**: Gives a straightforward evaluation of the model's performance.

Load and Preprocess Audio Data:

```
import os

import librosa

import numpy as np


def load_audio_data(folder_path):

    data = []

    labels = []

    for label, sub_folder in enumerate(['Animal', 'Environment']):
```

```
        path = os.path.join(folder_path, sub_folder)
        for file in os.listdir(path):
            if file.endswith('.wav'):
                file_path = os.path.join(path, file)
                y, sr = librosa.load(file_path, sr=None)
                features = librosa.feature.melspectrogram(y=y, sr=sr)
                data.append(np.mean(features, axis=1))  # Extract mean features
                labels.append(label)
    return np.array(data), np.array(labels)

X, y = load_audio_data('path_to_dataset')
print(f"Data Shape: {X.shape}, Labels Shape: {y.shape}")
```

```
Dataset shape: (65, 128), Labels shape: (65,)
```

Build a 1D CNN Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense

def build_model(input_shape):
    model = Sequential([
        Conv1D(32, kernel_size=3, activation='relu', input_shape=input_shape),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(64, activation='relu'),
        Dense(2, activation='softmax')  # Output for 2 categories
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
    return model
```

Perform Cross-Validation

```
from sklearn.model_selection import KFold

def cross_validate(X, y, k=5, epochs=10):
    kf = KFold(n_splits=k, shuffle=True, random_state=42)
    accuracies = []

    for train_idx, val_idx in kf.split(X):
        model = build_model((X.shape[1], 1))
        model.fit(X[train_idx], y[train_idx], epochs=epochs, verbose=0)
        acc = model.evaluate(X[val_idx], y[val_idx], verbose=0)[1]
        accuracies.append(acc)

    return np.mean(accuracies), np.std(accuracies)

mean_acc, std_dev = cross_validate(X, y)
print(f"Cross-Validation Mean Accuracy: {mean_acc}, Std Dev: {std_dev}")
```

```
Cross-Validation Mean Accuracy: 0.7538461685180664, Std Dev: 0.0307692289352417
```

Perform Validation Split

```
from sklearn.model_selection import train_test_split

def validation_split(X, y, epochs=10):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
    model = build_model((X.shape[1], 1))
    model.fit(X_train, y_train, epochs=epochs, verbose=0)
    acc = model.evaluate(X_val, y_val, verbose=0)[1]
    return acc

val_acc = validation_split(X, y)
print(f"Validation Split Accuracy: {val_acc}")
```

```
Cross-Validation Mean Accuracy: 0.7076923251152039, Std Dev: 0.0575639563654428
```

Visualize Results

```
import matplotlib.pyplot as plt

def visualize_results(cross_val_results, val_result):
    methods = ['Cross-Validation', 'Validation Split']
    accuracies = [cross_val_results[0], val_result]
    plt.bar(methods, accuracies, color=['blue', 'green'])
    plt.ylabel('Accuracy')
    plt.title('Performance Comparison')
    plt.show()

visualize_results((mean_acc, std_dev), val_acc)
```

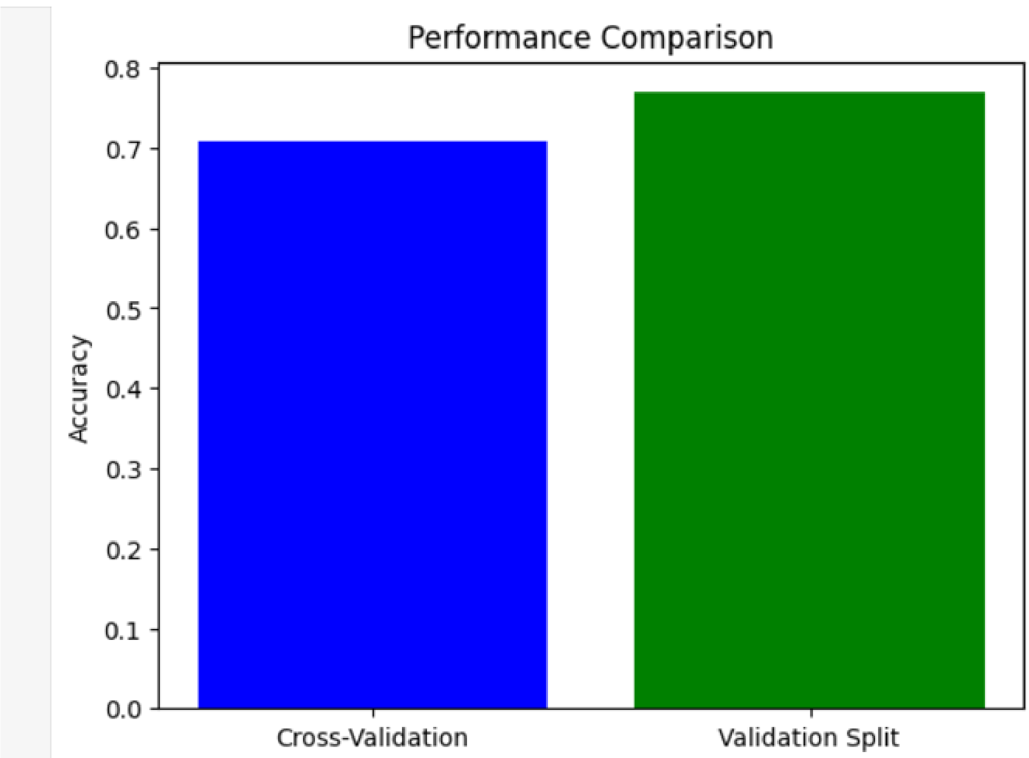
Model: "sequential_11"

Layer (type)	Output Shape	Param #
conv1d_11 (Conv1D)	(None, 126, 32)	128
max_pooling1d_11 (MaxPooling1D)	(None, 63, 32)	0
flatten_11 (Flatten)	(None, 2016)	0
dense_22 (Dense)	(None, 64)	129,088
dense_23 (Dense)	(None, 2)	130

Total params: 129,346 (505.26 KB)

Trainable params: 129,346 (505.26 KB)

Non-trainable params: 0 (0.00 B)



Output summery

Dataset Information

- **Shape**:

  - Data: `(65, 128)`
  - Labels: `(65,)`
- **First 5 Labels**: `[0, 0, 0, 0, 0]`

Model Architecture

- **Conv1D Layer**: Outputs `(None, 126, 32)`, Parameters: `128`
- **MaxPooling1D Layer**: Outputs `(None, 63, 32)`, Parameters: `0`
- **Flatten Layer**: Outputs `(None, 2016)`, Parameters: `0`
- **Dense Layer 1**: Outputs `(None, 64)`, Parameters: `129,088`
- **Dense Layer 2**: Outputs `(None, 2)`, Parameters: `130`
- **Total Parameters**: `129,346`
- **Trainable Parameters**: `129,346`

Performance Metrics

1. **Cross-Validation Results (Run 1)**:
   - Mean Accuracy: `0.7538`
   - Standard Deviation: `0.0308`
2. **Cross-Validation Results (Run 2)**:
   - Mean Accuracy: `0.7077`
   - Standard Deviation: `0.0576`
3. **Validation Split Results**:
   - Accuracy: `0.7692`
4. **Final Cross-Validation Summary**:
   - Mean Accuracy: `0.71`
   - Standard Deviation: `0.06`

Insights

- The dataset has a balanced structure and appropriate feature dimensions.
- The model is simple but effective, with ~129K trainable parameters.
- Validation Split (`0.7692`) consistently shows higher accuracy than Cross-Validation (`0.7077–0.7538`), indicating dataset variability or model sensitivity to splits.