Modified-Audio-Similarity-Search

October 20, 2022

0.1 Import libraries

[ ]:
```
# Python built-ins
import time
import os
import random

# math/vector maniuplation and plots
import numpy as np
import matplotlib.pyplot as plt

# image saving
from skimage import io

# audio processing library
import librosa
import librosa.display

# audio playback in notebook
import IPython.display as ipd

# Scikit Learn
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics.pairwise import cosine_similarity

# TensorFlow and Keras
import tensorflow as tf
from tensorflow import keras
```

0.2 Global Variables

[27]:
```
max_audio_duration = 2 # seconds
audio_samplerate = 22050 # Hz
n_fft = 4096 # number of ffts per window
fmin = 20 # min frequency to consider in mel cepstral coeffs
```

```
fmax = audio_samplerate // 2 # max frequency to consider in mel cepstral coeffs
input_image_size = (256, 256) # (width, height) pixels
image_value_type = np.uint16 # bit-depth of image, 16-bit seems to provide the best results
image_max_value = np.iinfo(image_value_type).max # used in scaling values 0-1
test_set_size = 0.2 # % of dataset to reserve for test set
trained_encoder_location = 'trained_encoders/ConvEncoder'
training_clip_paths = ['Test1', 'Test2', "Test3"]
inference_clip_paths = ['Test1', 'Test2', 'Test3']
```

0.3 Helper Methods

[28]:
```
def get_clip_paths(rel_paths):
    """
    Returns a list of full paths for all audio supported files under each path in rel_paths.
    """
    clip_paths = []
    for rel_path in rel_paths:
        audio_wav_dir = f'audio-data/{rel_path}'
        for root, dirs, files in os.walk(audio_wav_dir):
            for file in files:
                path = os.path.join(root, file)
                if path.endswith('.wav'):
                    clip_paths.append(path)
    return clip_paths
```

[29]:
```
def get_audio_samples(path, min_samples):
    """
    Returns list of samples for an audio file at path,
    """
    y, _ = librosa.load(path=path, mono=True, sr=audio_samplerate, offset = 0, duration = max_audio_duration)
    # check if needs padding, up to n_fft used in mel calcs
    if len(y) < min_samples:
        padding = min_samples - len(y)
        y = np.pad(y, (0, min_samples - len(y)), 'constant')
    return y
```

[30]:
```
def scale_minmax(X, max=image_max_value):
    """
    Normalizes values in range 0 - max.
    """
    X_min = X.min()
    X_max = X.max()
    X_diff = X_max - X_min
```

```
    return ((X - X_min)/X_diff) * max

def get_mel_spectrogram(y, sr, filename, save_file=False):
    """
    Returns mel spectrogram image data given audio data.
    """
    S = librosa.feature.melspectrogram(y=y,
                                       sr=sr,
                                       n_mels=input_image_size[1],
                                       n_fft=n_fft,
                                       hop_length=max(int(len(y)/
 ↪input_image_size[0]), 1),
                                       fmin = fmin,
                                       fmax = fmax)

    # convert to power-scale (decibels)
    img = librosa.power_to_db(S, ref=np.max).astype(np.float32)

    # discard extra mfcc columns, pad missing
    if img.shape[1] > input_image_size[0]:
        img = img[:, :input_image_size[0]]
    elif img.shape[1] < input_image_size[0]:
        img = np.pad(img, [(0,0), (0,input_image_size[0] - img.shape[1])],␣
 ↪'constant')

    # scale mel values to image values
    img = scale_minmax(img)

    # put low frequencies at the bottom in image (typical human readable format)
    img = np.flip(img, axis=0)

    img = img.astype(image_value_type)

    # save as PNG
    if save_file:
        print(f'Saving PNG: {filename}')
        io.imsave(filename.replace(".wav", ".png"), img)

    return img
```

0.4 Data Ingestion and Preprocessing

[31]:
```
def get_data(paths, save_images=False):
    """
    Get data from paths, prepared for training from. Optionally save Mel␣
 ↪Spectrogram image files.
```

```
    """
    audio_paths = get_clip_paths(paths)
    mels = []
    paths = []
    for path in audio_paths:
        paths.append(path)
        samples = get_audio_samples(path, n_fft)
        mel = get_mel_spectrogram(samples, audio_samplerate, path,␣
 ↪save_file=save_images)
        # we add a color channel, as the model will expect this in the input␣
 ↪shape, and append to dataset
        mels.append(mel.reshape(input_image_size[1], input_image_size[0], 1))

    # Scale values to 0-1
    X = np.array(mels)
    X = X / float(image_max_value)
    return paths, X
```

0.5 Train Test Split

[32]:
```
Y, X = get_data(training_clip_paths, save_images=False)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,␣
 ↪test_size=test_set_size, random_state=42)
X_train = np.reshape(X_train, (len(X_train), input_image_size[0],␣
 ↪input_image_size[1], 1))
X_test = np.reshape(X_test, (len(X_test), input_image_size[0],␣
 ↪input_image_size[1], 1))
```

0.6 Build Convolutional Autoencoder

The Keras functional API was used to create the convolutional autoencoder shown below. It comprises of a convolutional and deconvolutional encoder and decoder, which are both rather straightforward.

The encoder makes an embedding with the shape of (16,16,16) this represents the 75% of the orgianl data. The encoded embedding is used by the Decoder to simply reassemble the input. Adam optimisation and MSE loss are employed by the model.

[33]:
```
input_img = tf.keras.layers.
 ↪Input(shape=(input_image_size[0],input_image_size[1], 1))

# Encoder
x = tf.keras.layers.Conv2D(128, (3, 3), activation="relu",␣
 ↪padding="same")(input_img)
x = tf.keras.layers.MaxPooling2D((2, 2), padding="same")(x)
x = tf.keras.layers.Conv2D(64, (3, 3), activation="relu", padding="same")(x)
x = tf.keras.layers.MaxPooling2D((2, 2), padding="same")(x)
```

```python
x = tf.keras.layers.Conv2D(32, (3, 3), activation="relu", padding="same")(x)
x = tf.keras.layers.MaxPooling2D((2, 2), padding="same")(x)
x = tf.keras.layers.Conv2D(16, (3, 3), activation="relu", padding="same")(x)
encoded = tf.keras.layers.MaxPooling2D((2, 2), padding="same")(x)

# encoded.shape = (16,16,16)
# ~94% value compression (16*16*16)/(256*256)
# ~75% memory compression (16*16*16*4 bytes)/(256*256*1 bytes)

# Decoder
x = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',␣
 ↪padding='same')(encoded)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
x = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = tf.keras.layers.UpSampling2D((2, 2))(x)
decoded = tf.keras.layers.Conv2D(1, (3, 3), padding='same')(x) # no activation


# Autoencoder
autoencoder = tf.keras.models.Model(input_img, decoded)
autoencoder.compile(optimizer="adam", loss="mean_squared_error")
autoencoder.summary()
```

```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256, 256, 1)]     0

 conv2d_9 (Conv2D)           (None, 256, 256, 128)     1280

 max_pooling2d_4 (MaxPooling  (None, 128, 128, 128)    0
 2D)

 conv2d_10 (Conv2D)          (None, 128, 128, 64)      73792

 max_pooling2d_5 (MaxPooling  (None, 64, 64, 64)       0
 2D)

 conv2d_11 (Conv2D)          (None, 64, 64, 32)        18464

 max_pooling2d_6 (MaxPooling  (None, 32, 32, 32)       0
 2D)
```

```
 conv2d_12 (Conv2D)          (None, 32, 32, 16)        4624

 max_pooling2d_7 (MaxPooling  (None, 16, 16, 16)       0
 2D)

 conv2d_13 (Conv2D)          (None, 16, 16, 16)        2320

 up_sampling2d_4 (UpSampling  (None, 32, 32, 16)       0
 2D)

 conv2d_14 (Conv2D)          (None, 32, 32, 32)        4640

 up_sampling2d_5 (UpSampling  (None, 64, 64, 32)       0
 2D)

 conv2d_15 (Conv2D)          (None, 64, 64, 64)        18496

 up_sampling2d_6 (UpSampling  (None, 128, 128, 64)     0
 2D)

 conv2d_16 (Conv2D)          (None, 128, 128, 128)     73856

 up_sampling2d_7 (UpSampling  (None, 256, 256, 128)    0
 2D)

 conv2d_17 (Conv2D)          (None, 256, 256, 1)       1153

=================================================================
Total params: 198,625
Trainable params: 198,625
Non-trainable params: 0
_________________________________________________________________
```

0.7 Train Autoencoder

Even on a CPU, this model trains fairly quickly. The model is only supposed to train for 10 epochs, but it seems to converge quickly, so it's unclear whether training for longer would produce noticeably better results.

[34]:
```python
autoencoder.fit(X_train, X_train,
                epochs=10,
                batch_size=32,
                shuffle=True,
                validation_data=(X_test, X_test))
```

```
Epoch 1/10
1/1 [==============================] - 2s 2s/step - loss: 0.0882 - val_loss:
```

```
0.0781
Epoch 2/10
1/1 [==============================] - 1s 1s/step - loss: 0.0781 - val_loss:
0.0541
Epoch 3/10
1/1 [==============================] - 1s 1s/step - loss: 0.0541 - val_loss:
0.0221
Epoch 4/10
1/1 [==============================] - 1s 1s/step - loss: 0.0221 - val_loss:
0.1050
Epoch 5/10
1/1 [==============================] - 1s 1s/step - loss: 0.1050 - val_loss:
0.0254
Epoch 6/10
1/1 [==============================] - 1s 1s/step - loss: 0.0254 - val_loss:
0.0221
Epoch 7/10
1/1 [==============================] - 1s 1s/step - loss: 0.0221 - val_loss:
0.0332
Epoch 8/10
1/1 [==============================] - 1s 1s/step - loss: 0.0332 - val_loss:
0.0407
Epoch 9/10
1/1 [==============================] - 1s 1s/step - loss: 0.0407 - val_loss:
0.0435
Epoch 10/10
1/1 [==============================] - 1s 1s/step - loss: 0.0435 - val_loss:
0.0425
```

[34]:
```
<keras.callbacks.History at 0x7f405c108eb0>
```

0.8 Save Trained Encoder Model

[35]:
```
model = tf.keras.models.Model(inputs=autoencoder.inputs, outputs=autoencoder.
 ↪layers[8].output)
model.summary
model.save(trained_encoder_location)
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op while saving (showing 4 of 4). These functions will
```

not be directly callable after loading.

INFO:tensorflow:Assets written to: trained_encoders/ConvEncoder/assets

INFO:tensorflow:Assets written to: trained_encoders/ConvEncoder/assets

0.9 Inference Using Encoder, Similarity Database Searching

To construct embeddings on the complete audio dataset, the encoder is imported and utilised below. Then, a Cosine Similarity matrix is generated directly from the embeddings and cached in a dictionary keyed by the audio file path of the "similarity database." Based on the highest similarity values for a given audio file, the get similar audio method can then immediately fetch the n results most similar audio files from the database.

[36]:
```python
# load model and get audio data
start = time.time()
encoder = keras.models.load_model(trained_encoder_location)
all_paths, data = get_data(inference_clip_paths)
data = np.reshape(data, (len(data), input_image_size[0], input_image_size[1],␣
↪1))
```

```
WARNING:tensorflow:No training configuration found in save file, so the model
was *not* compiled. Compile it manually.

WARNING:tensorflow:No training configuration found in save file, so the model
was *not* compiled. Compile it manually.
```

[37]:
```python
# get encodings and and create similarity (cosine) database
start = time.time()

# create embeddings for all input data
encoded_data = encoder.predict(data)

encoding_time = round(time.time()-start, 2)
print(f'encoding time: {encoding_time:.2f}s, {encoding_time/len(data):.3f}s per␣
↪audio file.')

encoded_data = [np.ravel(e) for e in encoded_data]
similarity_matrix = cosine_similarity(encoded_data)
```

```
WARNING:tensorflow:6 out of the last 8 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f405064ee50>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
WARNING:tensorflow:6 out of the last 8 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x7f405064ee50>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid unnecessary
retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
1/1 [==============================] - 0s 151ms/step
encoding time: 0.18s, 0.030s per audio file.
```

[38]:
```
# create similarity database
similarity_database = {}
for P, S in zip(all_paths, similarity_matrix):
    similarity_database[P] = S
```

[39]:
```
def get_similar_audio(path, similarity_database, n_results):
    """
    Retrieve n_results most similar audio files to the given audio file at path␣
 ↪from the given similarity_database
    """
    n_results += 1
    all_paths = list(similarity_database.keys())
    ipd.display(ipd.Audio(path))
    # get similarity results for this audio
    similar = similarity_database[path]
    # get indexes of n_results highest values
    result_indexes = np.argpartition(similar, -n_results)[-n_results:]
    # build dictionary from results of paths and similarity scores
    results = {k:v for (k,v) in zip([all_paths[x] for x in result_indexes],␣
 ↪[similar[x] for x in result_indexes])}
    # eliminate self from results
    results = {k:v for k, v in results.items() if k != path}
    # get keys for results by value in reverse sorted order
    sorted_keys = sorted(results, key=results.__getitem__, reverse=True)

    #show results
    count = 1
    for k in sorted_keys:
        print(f'Result {count}: {k}, Similarity: {results[k]:.3f}')
        ipd.display(ipd.Audio(k))
        count += 1
```

0.10 Testing Similarity Search

The similarity search results are displayed below. We choose a random audio file for each of the n examples from the database, and then we return the n results most comparable audio recordings. One can listen to both the original audio file and the returning results using the inline audio players.

[40]:
```
# test returning results and check playback
n_examples = 1
n_results = 4
for i in range(n_examples):
    path = "audio-data/Test1/FC.wav" # select a random path from all_paths
    print(f'Finding similar sounds to {path}')
    get_similar_audio(path, similarity_database, n_results)
    print('\n\n-----------------------------------------------------------\n\n')
```

```
Finding similar sounds to audio-data/Test1/FC.wav

<IPython.lib.display.Audio object>

Result 1: audio-data/Test2/FC.wav, Similarity: 1.000

<IPython.lib.display.Audio object>

Result 2: audio-data/Test3/FC.wav, Similarity: 1.000

<IPython.lib.display.Audio object>

Result 3: audio-data/Test2/F.wav, Similarity: 0.733

<IPython.lib.display.Audio object>

Result 4: audio-data/Test3/F.wav, Similarity: 0.733

<IPython.lib.display.Audio object>



-----------------------------------------------------------
```