Voice Controlled Web Camera

Although the author and publisher have made every effort to ensure that the information in this writing was correct at press time, the author and publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

The goal of this small project is to create a voice controlled web camera. The idea is to allow user to pose freely and the camera captures his/her images when a certain word is said. Put it simply, it enables user to activate the camera without requiring him/her to touch any button (self-portrait).

In this example, the program recognizes three words: 'shoot', 'lighter', and 'darker'. 'shoot' makes the camera captures user's image. 'lighter' and 'darker' are meant to manage a lighting system (has not yet completed).

The project itself consists of two python scripts, *makemodel.py* and *main.py*. The first script, as the name implies, creates a neural network model that will be used by the second script to determine the word that has been said by the user. So, basically, it is a multi classification problem.

In order to build the model, a collection of of wav files that function as the training data need to be prepared. Here is the directory structure:

```
voice
  blank
   blank1.wav
   blank2.wav
   ...
  darker
   darker1.wav
   darker2.wav
   ...
  lighter
```

```
   lighter1.wav
   lighter2.wav
   ...
  shoot
   shoot1.wav
   shoot2.wav
   ...
```

The files in the *blank* directory represent signals when user says nothing.

Here are the scripts:

makemodel.py

```
import os
import numpy as np
import librosa
from scipy.io import wavfile
from scipy import signal
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization


sample_data = []
lbl_data = []

def get_file_list(directory):
    for dir_path, dir_names, file_names in os.walk(directory):
        for file_name in file_names:
            yield open(os.path.join(dir_path,file_name),'r',encoding='utf-8')

f_list = get_file_list('voice')

for f_path in f_list:
    samples, _ = librosa.load(f_path.name, sr = 8000)
    stft = librosa.stft(samples)
    stft = np.abs(stft)
    amp = librosa.amplitude_to_db(stft)
    sample_data.append(amp)
    if 'darker' in f_path.name:
        lbl_data.append(np.array([1,0,0,0]))
    elif 'lighter' in f_path.name:
        lbl_data.append(np.array([0,1,0,0]))
    elif 'shoot' in f_path.name:
        lbl_data.append(np.array([0,0,1,0]))
    else:
        lbl_data.append(np.array([0,0,0,1]))

sample_data = np.array(sample_data).reshape(len(sample_data),1025,16,1)
lbl_data = np.array(lbl_data)

model = Sequential()
model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(1025,16,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy',  optimizer='adam', metrics=['accuracy'])
model.fit(sample_data, lbl_data, batch_size=10, epochs=10)
model.save('model.h5')
```

Explanation:

```
samples, _ = librosa.load(f_path.name, sr = 8000)
stft = librosa.stft(samples)
stft = np.abs(stft)
amp = librosa.amplitude_to_db(stft)
sample_data.append(amp)
```

As you know, voice or sound, in digital form, consists of a sequence of samples. The raw voice data describes amplitude vs time. However, when dealing with this kind of project, we do not work with raw amplitude data. We also need to know the frequency components that exis in the signal. The function *librosa.stft* gets the job done. It converts raw data to a spectrogram that describes amplitude at a particular time and frequency. Simply put, one dimensional vectors are turned into two dimensional vectors. We treat spectograms just like we work with 2D images. As librosa produces complex numbers, we take the real part (remove the imaginary number) with *np.abs(stft)*. The amplitude is then transformed to decibels using *librosa.amplitude_to_db(stft)* so that the maximum value will be 0.

```
if 'darker' in f_path.name:
    lbl_data.append(np.array([1,0,0,0]))
elif 'lighter' in f_path.name:
    lbl_data.append(np.array([0,1,0,0]))
elif 'shoot' in f_path.name:
    lbl_data.append(np.array([0,0,1,0]))
else:
    lbl_data.append(np.array([0,0,0,1]))
```

Each of the sample voice data is labeled for the purpose of model building. All the files in the *darker* directory, for instance, is labeled *[1,0,0,0]*

```
model = Sequential()
model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(1025,16,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(4, activation='softmax'))

model.compile(loss='categorical_crossentropy',  optimizer='adam', metrics=['accuracy'])
model.fit(sample_data, lbl_data, batch_size=10, epochs=10)
model.save('model.h5')
```

The process to build the model. We train the model using the spectograms (*sample_data*) and the corresponding labels (*lbl_data*). As it is a multi classification problem, we use *categorical_crossentropy* as the loss function.

main.py

```
import cv2
import numpy as np
import librosa
import sounddevice as sd
from scipy.io.wavfile import write
from keras.models import load_model

filename = 'user.wav'
counter = 1
srate = 8000             #Sample rate
duration = 1             #Duration
model = load_model('model.h5')

while(True):
```

```
    recording = sd.rec(int(duration * srate), samplerate = srate, channels = 2)
    sd.wait()
    write(filename, srate, recording)

    samples, _ = librosa.load(filename, sr = 8000)
    stft = librosa.stft(samples)
    stft = np.abs(stft)
    amp = librosa.amplitude_to_db(stft)

    result = model.predict(np.array(amp).reshape(-1, 1025, 16, 1))
    print(result)

    if np.max(result) == result[0][0]:
        print('darker')
    elif np.max(result) == result[0][1]:
        print('lighter')
    elif np.max(result) == result[0][2]:
        #capture image using webcam
        print('shoot')
        cam = cv2.VideoCapture(0)
        img_name = 'user'+str(counter)+'.png'
        ret, frame = cam.read()
        cv2.imwrite(img_name, frame)
        cam.release()
        counter += 1
    else:
        print('blank')
```

Explanation:

```
recording = sd.rec(int(duration * srate), samplerate = srate, channels = 2)
sd.wait()
write(filename, srate, recording)
```

The recording process. Say one of the three words ('shoot', 'lighter', or 'darker'). User voice will be saved as a wav file (*user.wav*).

```
samples, _ = librosa.load(filename, sr = 8000)
stft = librosa.stft(samples)
stft = np.abs(stft)
amp = librosa.amplitude_to_db(stft)
```

The recorded user file (*user.wav*) is then loaded and processed so that it can be read by the model.

```
result = model.predict(np.array(amp).reshape(-1, 1025, 16, 1))
```

Using the model, user voice is categorized.

```
if np.max(result) == result[0][0]:
        print('darker')
elif np.max(result) == result[0][1]:
        print('lighter')
elif np.max(result) == result[0][2]:
        #capture image using webcam
        print('shoot')
        cam = cv2.VideoCapture(0)
        img_name = 'user'+str(counter)+'.png'
        ret, frame = cam.read()
        cv2.imwrite(img_name, frame)
        cam.release()
        counter += 1
else:
        print('blank')
```

A certain action is performed based on the user voice. If the user said 'shoot', an image will be captured.
