2022 AI보안연구센터 인턴 미팅

Research REPORT

22년 12월 12일

20170622

이종헌

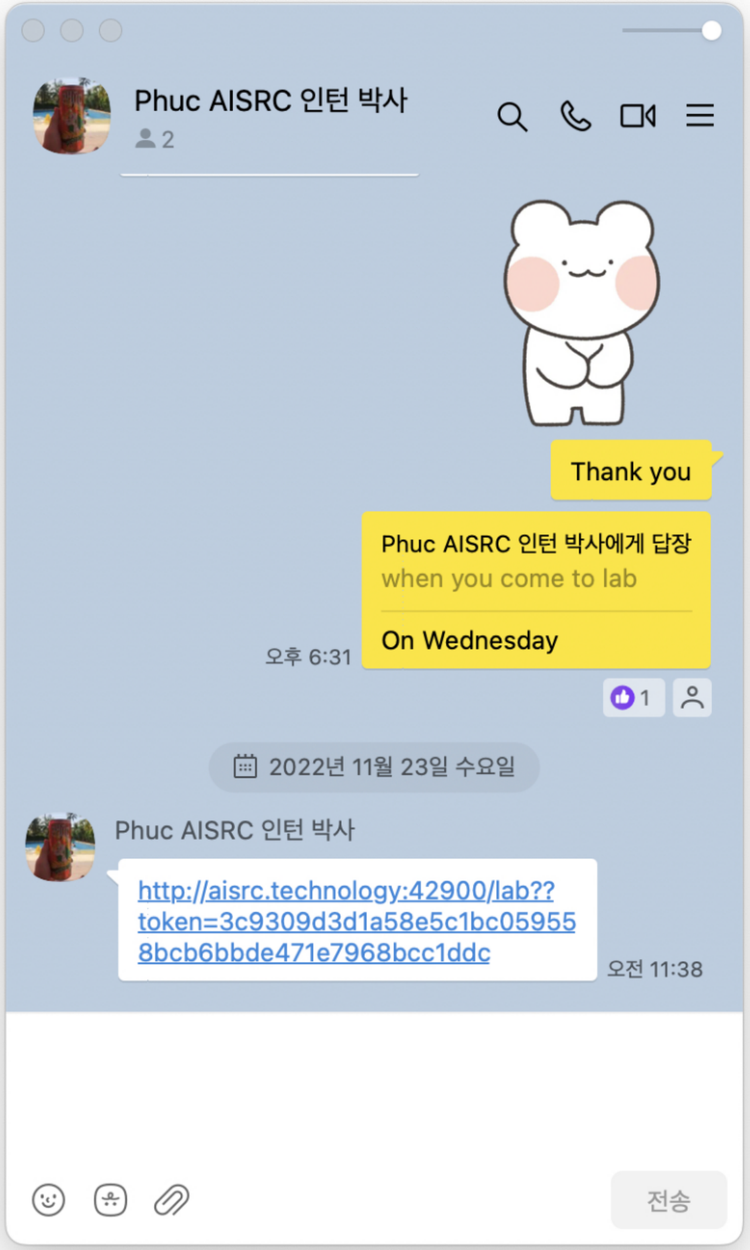
Task

Deepfake Voice 탐지의 이해 및
경량화를 통해 안드로이드 앱으로 구현

1. Deepfake Voice를
탐지하는 모델을 이해하고

2. 딥러닝 모델을 경량화하여

3. 안드로이드 앱으로 구현



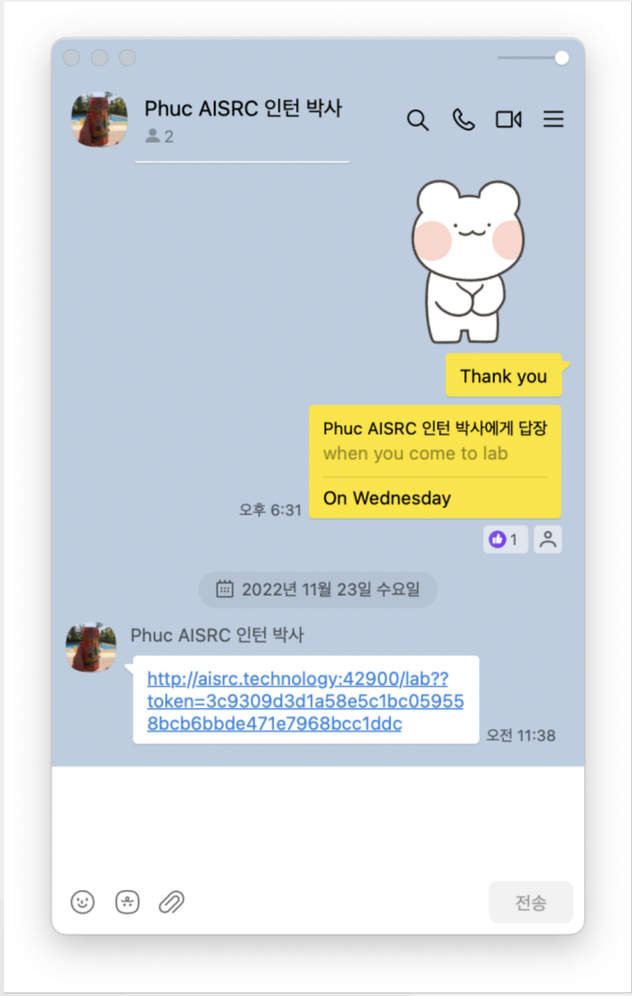
Task- This week Goal

새롭게 Phuc님이 만든 Deepfake Voice 탐지 모델을 안드로이드에서 구현해봐라

새로 받은 코드들을 분석완료

안드로이드에서 오디오를 받는 방법에
대해서 파악하고, 모델을 임포트 해보는 작업까지 완료

-> 이번주는 데이터 전처리를 안드로이드 환경에서 진행



데이터 전처리 - Python GPU 환경

Pad

input : audio data, sampling할 길이

Output : padded_x

(원하는 길이만큼 오디오 데이터를 자르기)

parse_ input

input : audio file path

(판별하고자하는 오디오 파일 경로)

Output : 전처리된 오디오 데이터

```python
def pad(x, max_len=64600):
    x_len = x.shape[0]
    if x_len >= max_len:
        return x[:max_len]
    # need to pad
    num_repeats = int(max_len / x_len)+1
    padded_x = np.tile(x, (1, num_repeats))[:, :max_len][0]
    return padded_x

def parse_input(file_path):
    cut = 64600  # take ~4 sec audio (64600 samples)
    X, fs = librosa.load(file_path, sr=16000)
    print(X,type(X))
    X_pad = pad(X, cut)
    x_inp = Tensor(X_pad)
    return x_inp.unsqueeze(0).to(device)
```

Functions - parse_input (데이터 전처리 과정)

parse_ input

input으로 audio file path를 받으면

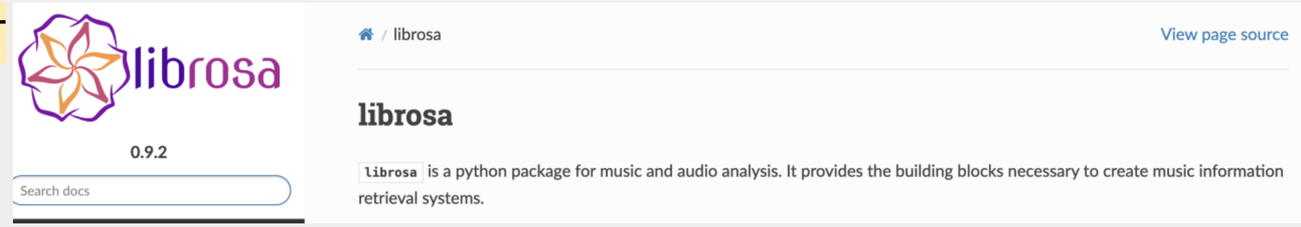
1. cut에 얼마만큼의 샘플을 기준으로 오디오를 자르고 싶은지 변수에 저장

현재는 64600 sample로 4초간의 오디오로 판단

2. librosa.load 함수를 통해

X에는 넘파이 형식으로 변환된 오디오 데이터를,

fs에는 Sample rate(초당 샘플링 횟수)를 저장한다.




```
def parse_input(file_path):
    cut = 64600  # take ~4 sec audio (64600 samples)
    X, fs = librosa.load(file_path, sr=16000)
    print(X,type(X))
    X_pad = pad(X, cut)
    x_inp = Tensor(X_pad)
    return x_inp.unsqueeze(0).to(device)
```

```
[0.00036621 0.00030518 0.00036621 ... 0.         0.         0.        ] <class 'numpy.ndarray'>
```

Functions - parse_input (데이터 전처리 과정)

parse_ input

3. pad 함수에 위에서 선언한 X와 cut을 파라미터로 넣어 오디오를 자르고
4. Tensor()를 통한 Tensor 형식으로 형변환
5. Tensor 형식의 데이터를 unsqueeze 한 후 결과로 리턴

```python
def parse_input(file_path):
    cut = 64600  # take ~4 sec audio (64600 samples)
    X, fs = librosa.load(file_path, sr=16000)
    print(X,type(X))
    X_pad = pad(X, cut)
    x_inp = Tensor(X_pad)
    return x_inp.unsqueeze(0).to(device)
```

```
[0.00036621 0.00030518 0.00036621 ... 0.         0.         0.        ] <class 'numpy.ndarray'>
tensor([ 0.0004,  0.0003,  0.0004,  ..., -0.0035,  0.0002, -0.0019]) <class 'torch.Tensor'> her
```

텐서(TENSOR) [원문 보기]

텐서(tensor)는 배열(array)이나 행렬(matrix)과 매우 유사한 특수한 자료구조입니다. PyTorch에서는 텐서를 사용하여 모델의 입력과 출력뿐만 아니라 모델의 매개변수를 부호화(encode)합니다.

GPU나 다른 연산 가속을 위한 특수한 하드웨어에서 실행할 수 있다는 점을 제외하면, 텐서는 NumPy의 ndarray와 매우 유사합니다.

```
>>> x = torch.tensor([1, 2, 3, 4])
>>> torch.unsqueeze(x, 0)
tensor([[ 1,  2,  3,  4]])
```

parse_input (데이터 전처리 과정) - JAVA

```java
public class MainActivity extends AppCompatActivity implements Runnable {
    private static final String TAG = MainActivity.class.getName();

    private Module module;
    private TextView mTextView;
    private Button mButton;

    private final static int REQUEST_RECORD_AUDIO = 13;
    private final static int AUDIO_LEN_IN_SECOND = 4;
    private final static int SAMPLE_RATE = 16000;
    private final static int RECORDING_LENGTH = SAMPLE_RATE * AUDIO_LEN_IN_SECOND;

    private final static String LOG_TAG = MainActivity.class.getSimpleName();

    private int mStart = 1;
    private HandlerThread mTimerThread;
    private Handler mTimerHandler;
    private Runnable mRunnable = new Runnable() {
        @Override
        public void run() {
            mTimerHandler.postDelayed(mRunnable, delayMillis: 1000);

            MainActivity.this.runOnUiThread(
                    () -> {
                        mButton.setText(String.format("Listening - %ds left", AUDIO_LEN_IN_SECOND - mStart));
                        mStart += 1;
                    });
        }
    };
```

1. cut에 얼마만큼의 샘플을 기준으로 오디오를 자르고 싶은지 변수에 저장
현재는 64600 sample로 4초간의 오디오, 16000의 Sampling Rate로 판단

int AUDIO_LEN_IN_SECOND :
몇 초간 녹음할 것인지 지정

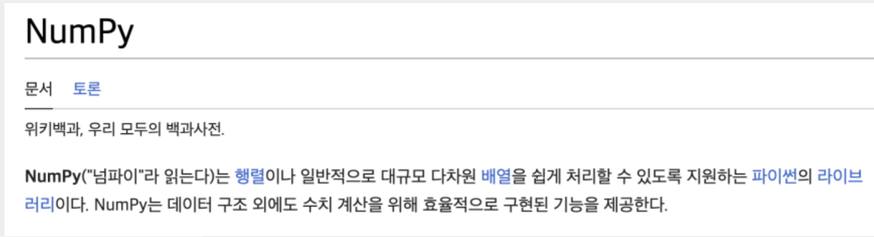
int SAMPLE_RATE :
샘플링(Sampling), 초당 몇 개의 디지털 신호를 뽑아낼지 지정

int RECORDING_LENGTH :
모델(Model)에 인풋(Input)으로 들어갈 배열의 길이.

parse_input (데이터 전처리 과정) - JAVA

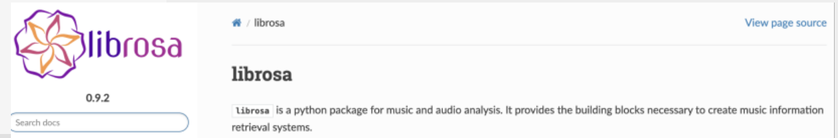
1. cut에 얼마만큼의 샘플을 기준으로 오디오를 자르고 싶은지 변수에 저장
현재는 64600 sample로 4초간의 오디오로 판단

2. librosa.load 함수를 통해
X에는 넘파이 형식으로 변환된 오디오 데이터를,
fs에는 Sample rate(초당 샘플링 횟수)를 저장한다.
-> 파이썬에서만 사용하는 Numpy 라이브러리인데
-> 이것을 Java에서 대체할 만한 라이브러리를 아직 못찾음




```
def parse_input(file_path):
    cut = 64600  # take ~4 sec audio (64600 samples)
    X, fs = librosa.load(file_path, sr=16000)
    print(X,type(X))
    X_pad = pad(X, cut)
    x_inp = Tensor(X_pad)
    return x_inp.unsqueeze(0).to(device)
```

```
[0.00036621 0.00030518 0.00036621 ... 0.         0.         0.        ] <class 'numpy.ndarray'>
```



데이터 전처리 과정 이슈 - JAVA

parse_ input, pad 함수를 변형하면 되는데
이때 사용되는 라이브러리가
librosa와 numpy이다.

NumPy("넘파이"라 읽는다)는 행렬이나 일반적으로 대규모 다차원 배열을 쉽게 처리할 수 있도록 지원하는 파이썬의 라이브러리이다.

librosa는 음악 및 오디오 분석을 위한 Python 패키지입니다.

-> 이번주는 두 라이브러리와 사용되지 않는 부분은 변환 완료
-> 언어가 다른 만큼 두개와 Java에서 대치되는 라이브러리를 탐색 및 시도

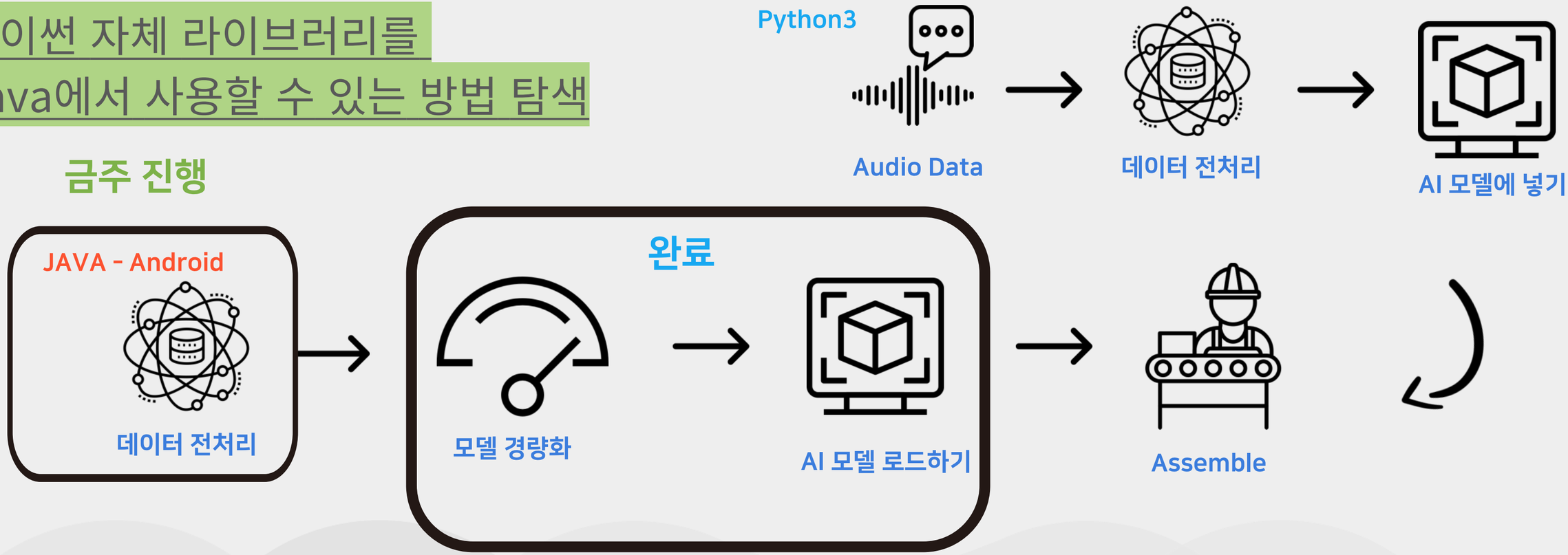
Next Week Task- 안드로이드에서 오디오 전처리 진행

완료된 작업 ->

모델 경량화, 안드로이드에 AI 모델 로드하기, 안드로이드에서 오디오 처리과정 분석

파이썬 자체 라이브러리가 필요하지 않은 데이터 전처리

파이썬 자체 라이브러리를
Java에서 사용할 수 있는 방법 탐색



Load ptl Pytorch Model on Android

Android Conversion

[7]:
```
import torch
from torch import Tensor
from torch.utils.mobile_optimizer import optimize_for_mobile
```

[8]:
```
class BTSDetect(torch.nn.Module):
    def __init__(self, model):
        super().__init__()
        self.model = model

    def forward(self, wavforms: Tensor):
        logits, _ = self.model(wavforms)
        _, pred = logits.max(dim=1)
        if pred:
            return 0 # bonafide
        else:
            return 1 # fake

model = BTSDetect(model)
```

[9]:
```
# # Apply quantization / script / optimize for motbile

quantized_model = torch.quantization.quantize_dynamic(
    model, qconfig_spec={torch.nn.Linear}, dtype=torch.qint8)
scripted_model = torch.jit.script(quantized_model)
optimized_model = optimize_for_mobile(scripted_model)


# Sanity check

print('Result:', optimized_model(x_inp))
optimized_model._save_for_lite_interpreter("btsdetect.ptl")
```

```
Result: 1
```

```
    public void run() {
        showTranslationResult(result);
        mButton.setEnabled(true);
        mButton.setText("Start");
    }
});


vate String recognize(float[] floatInputBuffer) {
if (module == null) {
    module = LiteModuleLoader.load(assetFilePath(getApplicationContext(), assetName: "btsdetect.ptl"));
}

double wav2vecinput[] = new double[RECORDING_LENGTH];
for (int n = 0; n < RECORDING_LENGTH; n++)
    wav2vecinput[n] = floatInputBuffer[n];

FloatBuffer inTensorBuffer = Tensor.allocateFloatBuffer(RECORDING_LENGTH);
for (double val : wav2vecinput)
    inTensorBuffer.put((float)val);

Tensor inTensor = Tensor.fromBlob(inTensorBuffer, new long[]{1, RECORDING_LENGTH});
final String result = module.forward(IValue.from(inTensor)).toStr();

return result;
```

Phuc이 작성한 Android Conversion 이라는 코드를 통해 "btsdetect.ptl" 이라는 Pytorch Model 생성 및 다운로드 -> Android에 Load

Functions - parse_input (데이터 전처리 과정)

parse_ input

input으로 audio file path를 받으면

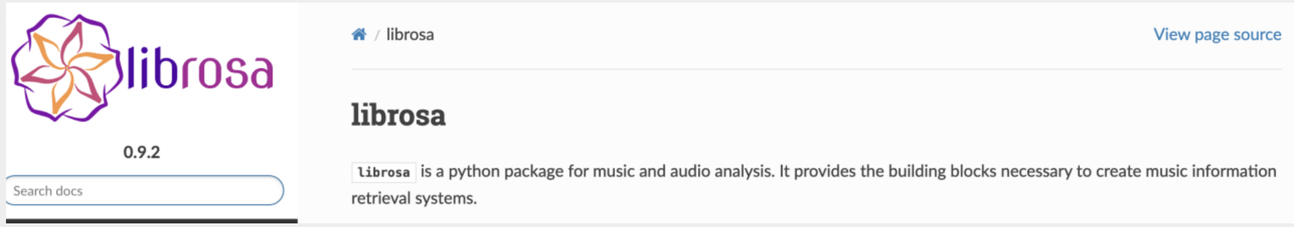
1. cut에 얼마만큼의 샘플을 기준으로 오디오를 자르고 싶은지 변수에 저장

현재는 64600 sample로 4초간의 오디오로 판단

2. librosa.load 함수를 통해

X에는 넘파이 형식으로 변환된 오디오 데이터를,

fs에는 Sample rate(초당 샘플링 횟수)를 저장한다.




```
def parse_input(file_path):
    cut = 64600  # take ~4 sec audio (64600 samples)
    X, fs = librosa.load(file_path, sr=16000)
    print(X,type(X))
    X_pad = pad(X, cut)
    x_inp = Tensor(X_pad)
    return x_inp.unsqueeze(0).to(device)
```

```
[0.00036621 0.00030518 0.00036621 ... 0.         0.         0.        ] <class 'numpy.ndarray'>
```

Functions - parse_input (데이터 전처리 과정)

parse_ input

3. pad 함수에 위에서 선언한 X와 cut을 파라미터로 넣어 오디오를 자르고
4. Tensor()를 통한 Tensor 형식으로 형변환
5. Tensor 형식의 데이터를 unsqueeze 한 후 결과로 리턴

```python
def parse_input(file_path):
    cut = 64600  # take ~4 sec audio (64600 samples)
    X, fs = librosa.load(file_path, sr=16000)
    print(X,type(X))
    X_pad = pad(X, cut)
    x_inp = Tensor(X_pad)
    return x_inp.unsqueeze(0).to(device)
```

```
[0.00036621 0.00030518 0.00036621 ... 0.         0.         0.        ] <class 'numpy.ndarray'>
tensor([ 0.0004,  0.0003,  0.0004,  ..., -0.0035,  0.0002, -0.0019]) <class 'torch.Tensor'> her
```

텐서(TENSOR) [원문 보기]

텐서(tensor)는 배열(array)이나 행렬(matrix)과 매우 유사한 특수한 자료구조입니다. PyTorch에서는 텐서를 사용하여 모델의 입력과 출력뿐만 아니라 모델의 매개변수를 부호화(encode)합니다.

GPU나 다른 연산 가속을 위한 특수한 하드웨어에서 실행할 수 있다는 점을 제외하면, 텐서는 NumPy의 ndarray와 매우 유사합니다.

```
>>> x = torch.tensor([1, 2, 3, 4])
>>> torch.unsqueeze(x, 0)
tensor([[ 1,  2,  3,  4]])
```

load model - 모델을 로드하기

load model

1. config_path에 yaml 모델 설정파일 주소를 저장
2. model_path에 학습이 완료된 pth 형식의 모델 파일 주소를 저장
3. yaml 라이브러리를 통해 모델 설정 파일 불러와서 parser1 변수에 저장
4. RawNet 라이브러리를 통해 yaml 설정파일을 토대로 모델을 로드해서 model 변수에 저장

```
config_path = "model_config_RawNet.yaml"
model_path = "models/model_LA_CCE_100_64_0.0001/epoch_71.pth"
device = 'cpu'
with open(config_path, 'r') as f_yaml:
    parser1 = yaml.safe_load(f_yaml)

model = RawNet(parser1['model'], device).to(device)
model.load_state_dict(torch.load(model_path,map_location=device))
```

```
<All keys matched successfully>
```

load model - 모델을 로드하기

load model

1. config_path에 yaml 모델 설정파일 주소를 저장
2. model_path에 학습이 완료된 pth 형식의 모델 파일 주소를 저장
3. yaml 라이브러리를 통해 모델 설정 파일 불러와서 parser1 변수에 저장
4. RawNet 라이브러리를 통해 yaml 설정파일을 토대로 모델을 로드해서 model 변수에 저장

```
config_path = "model_config_RawNet.yaml"
model_path = "models/model_LA_CCE_100_64_0.0001/epoch_71.pth"
device = 'cpu'
with open(config_path, 'r') as f_yaml:
    parser1 = yaml.safe_load(f_yaml)

model = RawNet(parser1['model'], device).to(device)
model.load_state_dict(torch.load(model_path,map_location=device))
```

```
<All keys matched successfully>
```

모델 실행하기

Implement 모델

1. model.eval()를 통해 모듈을 evaluation mode로 전환
2. wav변수에 판별하고 싶은 오디오 파일 주소 저장
3. parse_input(wav)를 통해 오디오를 전처리하고 결과를 x_inp
4. x_inp 를 파라미터로 모델에 넣어서 결과값을 도출
5. 도출된 결과를 real인지 fake인지 Convert함

```
model.eval()
# Load the audio file that we need to check whether it is fake voice or not.
wav = "/root/dataset/Dataset/ASVspoof/LA/ASVspoof2019_LA_eval/flac/LA_E_5085671.flac"

x_inp = parse_input(wav)
print(x_inp)
# use the model to calculate the prediction value
# out, _ = model(x_inp)
out, _ = model(x_inp)

# Convert into readable format
_, pred = out.max(dim=1)
if pred:
    print("bonafide") #real voice
else:
    print("spoof") #fake voice
```

Android Conversion

Phuc님이 작성하신 Android Conversion code
결과로 btsdetect.ptl 모델 파일 하나를 리턴한다.

Android Conversion

[7]:
```
import torch
from torch import Tensor
from torch.utils.mobile_optimizer import optimize_for_mobile
```

[8]:
```
class BTSDetect(torch.nn.Module):
    def __init__(self, model):
        super().__init__()
        self.model = model

    def forward(self, wavforms: Tensor):
        logits, _ = self.model(wavforms)
        _, pred = logits.max(dim=1)
        if pred:
            return 0 # bonafide
        else:
            return 1 # fake

model = BTSDetect(model)
```

[9]:
```
# # Apply quantization / script / optimize for motbile

quantized_model = torch.quantization.quantize_dynamic(
    model, qconfig_spec={torch.nn.Linear}, dtype=torch.qint8)
scripted_model = torch.jit.script(quantized_model)
optimized_model = optimize_for_mobile(scripted_model)


# Sanity check

print('Result:', optimized_model(x_inp))
optimized_model._save_for_lite_interpreter("btsdetect.ptl")
```

```
Result: 1
```

reference

YAML이란?

발행: 2021년 6월 18일 •

URL 복사

YAML은 구성 파일 작성에 자주 사용되는 데이터 직렬화 언어입니다. YAML을 *yet another markup language*로 생각하는 사람도 있고, *YAML ain't markup language*(재귀 약어)로 생각하는 사람도 있습니다. 후자는 YAML이 문서가 아닌 데이터용임을 강조하는 말입니다.

YAML은 사람이 읽을 수 있고 이해하기 쉬워 프로그래밍 언어 중에서도 인기가 높습니다. 또한 다른 프로그래밍 언어와 함께 사용할 수도 있습니다. YAML은 그 유연성과 접근성으로 인해 Ansible 자동화 툴에서 Ansible Playbook 형태로 자동화 프로세스를 생성하는 데 사용됩니다.

Demo.ipynb | model_config_RawNet.yam

```yaml
optimizer: Adam
amsgrad: 1   #for adam optim



#model-related
model:
  nb_samp: 64600
  first_conv: 1024   # no. of filter coefficients
  in_channels: 1
  filts: [20, [20, 20], [20, 128], [128, 128]] # no. of filters channel in residual blocks
  blocks: [2, 4]
  nb_fc_node: 1088 # equal to teacher size (1024 rawnet2 + 64 bio)
  gru_node: 1024
  nb_gru_layer: 1
  nb_classes: 2
  is_light: True
```

reference

eval() [SOURCE]

Sets the module in evaluation mode.

This has any effect only on certain modules. See documentations of particular modules for details of their behaviors in training/evaluation mode, if they are affected, e.g. `Dropout`, `BatchNorm`, etc.

This is equivalent with `self.train(False)`.

See Locally disabling gradient computation for a comparison between *.eval()* and several similar mechanisms that may be confused with it.

Returns:

self

Return type:

Module

Note ≡ model.train( ) & model.eval( )

- **`model.train()`**: 훈련 데이터셋에 사용하며 모델 훈련이 진행될 것임을 알립니다. 이때 드롭아웃(dropout)이 활성화됩니다.
- **`model.eval()`**: 모델을 평가할 때는 모든 노드를 사용하겠다는 의미로 검증과 테스트 데이터셋에 사용합니다.