Kapre

Release 2017

Aug 07, 2023

Kapre

1 Why Kapre? 3
- 1.1 vs. Pre-computation . . . 3
- 1.2 vs. Your own implementation . . . 3

2 Workflow with Kapre 5

3 Installation 7

4 Example 9

5 Citation 11

6 Contribution 13
- 6.1 Examples . . . 13
- 6.2 time_frequency . . . 16
- 6.3 signal . . . 22
- 6.4 composed . . . 25
- 6.5 backend . . . 32
- 6.6 time_frequency_tflite . . . 34
- 6.7 Release Note . . . 37

Python Module Index 39

Index 41

Keras Audio Preprocessors - compute STFT, InverseSTFT, Melspectrogram, and others on GPU real-time.

Tested on Python 3.6, and 3.7.

CHAPTER 1

Why Kapre?

1.1 vs. Pre-computation

- You can optimize DSP parameters
- Your model deployment becomes much simpler and consistent.
- Your code and model has less dependencies

1.2 vs. Your own implementation

- Quick and easy!
- Consistent with 1D/2D tensorflow batch shapes
- Data format agnostic (*channels_first* and *channels_last*)
- Less error prone - Kapre layers are tested against Librosa (stft, decibel, etc) - which is (trust me) *trickier* than you think.
- Kapre layers have some extended APIs from the default *tf.signals* implementation such as.. - A perfectly invertible *STFT* and *InverseSTFT* pair - Mel-spectrogram with more options
- Reproducibility - Kapre is available on pip with versioning

CHAPTER 2

Workflow with Kapre

1. Preprocess your audio dataset. Resample the audio to the right sampling rate and store the audio signals (waveforms).
2. In your ML model, add Kapre layer e.g. *kapre.time_frequency.STFT()* as the first layer of the model.
3. The data loader simply loads audio signals and feed them into the model
4. In your hyperparameter search, include DSP parameters like *n_fft* to boost the performance.
5. When deploying the final model, all you need to remember is the sampling rate of the signal. No dependency or preprocessing!

CHAPTER 3

Installation

```
pip install kapre
```

CHAPTER 4

Example

See the Jupyter notebook at the example folder

CHAPTER 5

Citation

Please cite this paper if you use Kapre for your work.

```
@inproceedings{choi2017kapre,
  title={Kapre: On-GPU Audio Preprocessing Layers for a Quick Implementation of Deep
↪Neural Network Models with Keras},
  author={Choi, Keunwoo and Joo, Deokjin and Kim, Juho},
  booktitle={Machine Learning for Music Discovery Workshop at 34th International
↪Conference on Machine Learning},
  year={2017},
  organization={ICML}
}
```

CHAPTER 6

Contribution

Visit github.com/keunwoochoi/kapre and chat with us :)

6.1 Examples

We provide fully functioning code snippets here. More detailed examples are under documentations of all the layers and functions.

6.1.1 How To Import

```
import kapre  # to import the whole library
from kapre import (  # `time_frequency` layers can be directly imported from `kapre`
    STFT,
    InverseSTFT,
    Magnitude,
    Phase,
    MagnitudeToDecibel,
    ApplyFilterbank,
    Delta,
    ConcatenateFrequencyMap,
)
from kapre import (  # `signal` layers can be also directly imported from kapre
    Frame,
    Energy,
    MuLawEncoding,
    MuLawDecoding,
    LogmelToMFCC,
)
# from kapre import backend  # we can do this, but `backend` might be a too general␣
↪name
import kapre.backend  # for namespace sanity, you might prefer this
```

(continues on next page)

(continued from previous page)

```
from kapre import backend as kapre_backend  # or maybe this
from kapre.composed import (  # function names in `composed` are purposefully verbose.
    get_stft_magnitude_layer,
    get_melspectrogram_layer,
    get_log_frequency_spectrogram_layer,
    get_perfectly_reconstructing_stft_istft,
    get_stft_mag_phase,
    get_frequency_aware_conv2d,
)
```

6.1.2 Use STFT Magnitude

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from kapre import STFT, Magnitude, MagnitudeToDecibel

sampling_rate = 16000  # sampling rate of your input audio
duration = 20.0  # duration of the audio
num_channel = 2  # number of channels of the audio
input_shape = (int(sampling_rate * duration), num_channel)  # let's follow `channels_
↪last` convention

model = Sequential()
model.add(STFT(n_fft=2048, win_length=2018, hop_length=1024,
               window_name='hann_window', pad_end=False,
               input_data_format='channels_last', output_data_format='channels_last',
               input_shape=input_shape))  # complex64
model.add(Magnitude())   # float32
model.add(MagnitudeToDecibel())  # float32 but in decibel scale
model.summary()  # this would be an "audio frontend" of your model
"""
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
stft (STFT)                  (None, 311, 1025, 2)      0
_________________________________________________________________
magnitude (Magnitude)        (None, 311, 1025, 2)      0
_________________________________________________________________
magnitude_to_decibel (Magnit (None, 311, 1025, 2)      0
=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
"""
# A 20-second stereo audio signal is converted to a (311, 1025, 2) tensor.

# Now, you can add your own model. For example, let's add ResNet50
# with global average pooling, no pre-trained weights,
# and for a 10-class classification.

model.add(
    tf.keras.applications.ResNet50(
        include_top=True, weights=None, input_tensor=None,
```

(continues on next page)

(continued from previous page)

```
        input_shape=(311, 1025, 2), pooling='avg', classes=10
    )
)

model.summary()
"""
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
stft (STFT)                  (None, 311, 1025, 2)      0
_________________________________________________________________
magnitude (Magnitude)        (None, 311, 1025, 2)      0
_________________________________________________________________
magnitude_to_decibel (Magnit (None, 311, 1025, 2)      0
_________________________________________________________________
resnet50 (Functional)        (None, 10)                23605066
=================================================================
Total params: 23,605,066
Trainable params: 23,551,946
Non-trainable params: 53,120
_________________________________________________________________
"""
```

6.1.3 Use STFT Magnitude – a lazy version

```
from tensorflow.keras.models import Sequential
from kapre.composed import get_stft_magnitude_layer

sampling_rate = 16000  # sampling rate of your input audio
duration = 20.0  # duration of the audio
num_channel = 2  # number of channels of the audio
input_shape = (int(sampling_rate * duration), num_channel)  # let's follow `channels_
↪last` convention

model = Sequential(get_stft_magnitude_layer(input_shape=input_shape, return_
↪decibel=True))

model.summary()  # this lazy version provides an abstraction view of stft_magnitude
"""
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
stft_magnitude (Sequential)  (None, 622, 1025, 2)      0
=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
"""
# Here, a 20-second stereo audio signal is converted to a (622, 1025, 2) tensor.
# x2 more temporal frames compared to the example above because we didn't set hop_
↪length here,
# and that means it's set to a 25% hop length, not 50% as above.
```

(continues on next page)

(continued from previous page)

```
model.layers[0].summary()  # let's deep dive one level
"""
Model: "stft_magnitude"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
stft (STFT)                  (None, 622, 1025, 2)      0
_________________________________________________________________
magnitude (Magnitude)        (None, 622, 1025, 2)      0
_________________________________________________________________
magnitude_to_decibel (Magnit (None, 622, 1025, 2)      0
=================================================================
Total params: 0
Trainable params: 0
Non-trainable params: 0
_________________________________________________________________
"""
```

6.2 time_frequency

Time-frequency Keras layers.

This module has low-level implementations of some popular time-frequency operations such as STFT and inverse STFT. We're using these layers to compose layers in *kapre.composed* where more high-level and popular layers such as melspectrogram layer are provided. You should go check it out!

Note: Why time-frequency representation?

Every representation (STFT, melspectrogram, etc) has something in common - they're all 2D representations (time, frequency-ish) of audio signals. They're helpful because they decompose an audio signal, which is a simultaneous mixture of a lot of frequency components into different frequency bins. They have spatial property; the frequency bins are *sorted*, so frequency bins nearby has represent only slightly different frequency components. The frequency decomposition is also what's happening during human auditory perception through cochlea.

Which representation to use as input?

For a quick summary, check out my tutorial paper, A Tutorial on Deep Learning for Music Information Retrieval.

---

class kapre.time_frequency.**STFT**(*n_fft=2048*, *win_length=None*, *hop_length=None*, *window_name=None*, *pad_begin=False*, *pad_end=False*, *input_data_format='default'*, *output_data_format='default'*, ***kwargs*)

A Short-time Fourier transform layer.

It uses *tf.signal.stft* to compute complex STFT. Additionally, it reshapes the output to be a proper 2D batch.

If *output_data_format == 'channels_last'*, the output shape is (batch, time, freq, channel) If *output_data_format == 'channels_first'*, the output shape is (batch, channel, time, freq)

Parameters

- **n_fft** (*int*) – Number of FFTs. Defaults to *2048*
- **win_length** (*int or None*) – Window length in sample. Defaults to *n_fft*.

- **hop_length** (*int or None*) – Hop length in sample between analysis windows. Defaults to *win_length // 4* following Librosa.
- **window_name** (*str or None*) – *Name* of *tf.signal* function that returns a 1D tensor window that is used in analysis. Defaults to *hann_window* which uses *tf.signal.hann_window*. Window availability depends on Tensorflow version. More details are at *kapre.backend.get_window()*.
- **pad_begin** (*bool*) – Whether to pad with zeros along time axis (length: win_length - hop_length). Defaults to *False*.
- **pad_end** (*bool*) – Whether to pad with zeros at the finishing end of the signal.
- **input_data_format** (*str*) – the audio data format of input waveform batch. *'channels_last'* if it's *(batch, time, channels)* and *'channels_first'* if it's *(batch, channels, time)*. Defaults to the setting of your Keras configuration. (*tf.keras.backend.image_data_format()*)
- **output_data_format** (*str*) – The data format of output STFT. *'channels_last'* if you want *(batch, time, frequency, channels)* and *'channels_first'* if you want *(batch, channels, time, frequency)* Defaults to the setting of your Keras configuration. (*tf.keras.backend.image_data_format()*)
- ****kwargs** – Keyword args for the parent keras layer (e.g., *name*)

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.STFT(n_fft=1024, hop_length=512, input_shape=input_shape))
# now the shape is (batch, n_frame=3, n_freq=513, ch=1)
# and the dtype is complex
```

call(*x*)

Compute STFT of the input signal. If the *time* axis is not the last axis of *x*, it should be transposed first.

Parameters **x** (float *Tensor*) – batch of audio signals, (batch, ch, time) or (batch, time, ch) based on input_data_format

Returns A STFT representation of x in a 2D batch shape. *complex64* if *x* is *float32*, *complex128* if *x* is *float64*. Its shape is (batch, time, freq, ch) or (batch. ch, time, freq) depending on *output_data_format* and *time* is the number of frames, which is *((len_src + (win_length - hop_length) / hop_length) // win_length)* if *pad_end* is *True*. *freq* is the number of fft unique bins, which is *n_fft // 2 + 1* (the unique components of the FFT).

Return type (complex *Tensor*)

class kapre.time_frequency.**InverseSTFT**(*n_fft=2048*, *win_length=None*, *hop_length=None*, *forward_window_name=None*, *input_data_format='default'*, *output_data_format='default'*, ***kwargs*)

An inverse-STFT layer.

If *output_data_format == 'channels_last'*, the output shape is (batch, time, channel) If *output_data_format == 'channels_first'*, the output shape is (batch, channel, time)

Note that the result of inverse STFT could be longer than the original signal due to the padding. Do check the size of the result by yourself and trim it if needed.

Parameters

- **n_fft** (*int*) – Number of FFTs. Defaults to *2048*

- **win_length** (*int* or *None*) – Window length in sample. Defaults to *n_fft*.
- **hop_length** (*int* or *None*) – Hop length in sample between analysis windows. Defaults to *n_fft // 4* following Librosa.
- **forward_window_name** (*str or None*) – *Name* of *tf.signal* function that *was* used in the forward STFT. Defaults to *hann_window*, assuming *tf.signal.hann_window* was used. Window availability depends on Tensorflow version. More details are at *kapre.backend.get_window()*.
- **input_data_format** (*str*) – the data format of input STFT batch *'channels_last'* if you want *(batch, time, frequency, channels) 'channels_first'* if you want *(batch, channels, time, frequency)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **output_data_format** (*str*) – the audio data format of output waveform batch. *'channels_last'* if it's *(batch, time, channels) 'channels_first'* if it's *(batch, channels, time)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- ****kwargs** – Keyword args for the parent keras layer (e.g., *name*)

Example

```
input_shape = (3, 513, 1)  # 3 frames, 513 frequency bins, 1 channel
# and input dtype is complex
model = Sequential()
model.add(kapre.InverseSTFT(n_fft=1024, hop_length=512, input_shape=input_shape))
# now the shape is (batch, time=2048, ch=1)
```

call(*x*)

Compute inverse STFT of the input STFT.

Parameters **x** (complex *Tensor*) – batch of STFTs, (batch, ch, time, freq) or (batch, time, freq, ch) depending on *input_data_format*

Returns audio signals of x. Shape: 1D batch shape. I.e., (batch, time, ch) or (batch, ch, time) depending on *output_data_format*

Return type (*float*)

class kapre.time_frequency.**Magnitude**(**args*, ***kwargs*)

Compute the magnitude of the complex input, resulting in a float tensor

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.STFT(n_fft=1024, hop_length=512, input_shape=input_shape))
mode.add(Magnitude())
# now the shape is (batch, n_frame=3, n_freq=513, ch=1) and dtype is float
```

call(*x*)

Parameters **x** (complex *Tensor*) – input complex tensor

Returns magnitude of *x*

Return type (float *Tensor*)

class kapre.time_frequency.**Phase**(*approx_atan_accuracy=None*, ***kwargs*)

Compute the phase of the complex input in radian, resulting in a float tensor

Includes option to use approximate phase algorithm this will return the same results as the PhaseTflite layer (the tflite compatible layer).

Parameters **approx_atan_accuracy** (*int*) – if *None* will use tf.math.angle() to calculate the phase accurately. If an *int* this is the number of iterations to calculate the approximate atan() using a tflite compatible method. the higher the number the more accurate e.g. approx_atan_accuracy=29000. You may want to experiment with adjusting this number: trading off accuracy with inference speed.

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.STFT(n_fft=1024, hop_length=512, input_shape=input_shape))
model.add(Phase())
# now the shape is (batch, n_frame=3, n_freq=513, ch=1) and dtype is float
```

call(*x*)

Parameters **x** (complex *Tensor*) – input complex tensor

Returns phase of *x* (Radian)

Return type (float *Tensor*)

class kapre.time_frequency.**MagnitudeToDecibel**(*ref_value=1.0*, *amin=1e-05*, *dynamic_range=80.0*, ***kwargs*)

A class that wraps *backend.magnitude_to_decibel* to compute decibel of the input magnitude.

Parameters

- **ref_value** (*float*) – an input value that would become 0 dB in the result. For spectrogram magnitudes, ref_value=1.0 usually make the decibel-scaled output to be around zero if the input audio was in [-1, 1].
- **amin** (*float*) – the noise floor of the input. An input that is smaller than *amin*, it's converted to ‘amin.
- **dynamic_range** (*float*) – range of the resulting value. E.g., if the maximum magnitude is 30 dB, the noise floor of the output would become (30 - dynamic_range) dB

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.STFT(n_fft=1024, hop_length=512, input_shape=input_shape))
model.add(Magnitude())
model.add(MagnitudeToDecibel())
# now the shape is (batch, n_frame=3, n_freq=513, ch=1) and dtype is float
```

call(*x*)

Parameters **x** (*Tensor*) – float tensor. Can be batch or not. Something like magnitude of STFT.

Returns decibel-scaled float tensor of *x*.

Return type (*Tensor*)

class kapre.time_frequency.**ApplyFilterbank**(*type*, *filterbank_kwargs*, *data_format='default'*, ***kwargs*)

Apply a filterbank to the input spectrograms.

Parameters

- **filterbank** (*Tensor*) – filterbank tensor in a shape of (n_freq, n_filterbanks)
- **data_format** (*str*) – specifies the data format of batch input/output
- ****kwargs** – Keyword args for the parent keras layer (e.g., *name*)

Example

```
input_shape = (2048, 1)  # mono signal
n_fft = 1024
n_hop = n_fft // 2
kwargs = {
    'sample_rate': 22050,
    'n_freq': n_fft // 2 + 1,
    'n_mels': 128,
    'f_min': 0.0,
    'f_max': 8000,
}
model = Sequential()
model.add(kapre.STFT(n_fft=n_fft, hop_length=n_hop, input_shape=input_shape))
model.add(Magnitude())
# (batch, n_frame=3, n_freq=n_fft // 2 + 1, ch=1) and dtype is float
model.add(ApplyFilterbank(type='mel', filterbank_kwargs=kwargs))
# (batch, n_frame=3, n_mels=128, ch=1)
```

call(*x*)

Apply filterbank to *x*.

Parameters **x** (*Tensor*) – float tensor in 2D batch shape.

class kapre.time_frequency.**Delta**(*win_length=5*, *mode='symmetric'*, *data_format='default'*, ***kwargs*)

Calculates delta, a local estimate of the derivative along time axis. See torchaudio.functional.compute_deltas or librosa.feature.delta for more details.

Parameters

- **win_length** (*int*) – Window length of the derivative estimation. Defaults to 5
- **mode** (*str*) – Specifies pad mode of *tf.pad*. Case-insensitive. Defaults to 'symmetric'. Can be 'symmetric', 'reflect', 'constant', or whatever *tf.pad* supports.

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.STFT(n_fft=1024, hop_length=512, input_shape=input_shape))
model.add(kapre.Magnitude())
model.add(Delta())
# (batch, n_frame=3, n_freq=513, ch=1) and dtype is float
```

call(*x*)

Parameters **x** (*Tensor*) – a 2d batch (b, t, f, ch) or (b, ch, t, f)

Returns A tensor with the same shape as input data.

Return type (*Tensor*)

class kapre.time_frequency.**ConcatenateFrequencyMap**(*data_format='default'*, ***kwargs*)

Addes a frequency information channel to spectrograms.

The added frequency channel (=frequency map) has a linearly increasing values from 0.0 to 1.0, indicating the normalize frequency of a time-frequency bin. This layer can be applied to input audio spectrograms or any feature maps so that the following layers can be conditioned on the frequency. (Imagine something like positional encoding in NLP but the position is on frequency axis).

A combination of *ConcatenateFrequencyMap* and *Conv2D* is known as frequency-aware convolution (see References). For your convenience, such a layer is supported by *karep.composed.get_frequency_aware_conv2d()*.

Parameters

- **data_format** (*str*) – specifies the data format of batch input/output.
- ****kwargs** – Keyword args for the parent keras layer (e.g., *name*)

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.STFT(n_fft=1024, hop_length=512, input_shape=input_shape))
model.add(kapre.Magnitude())
# (batch, n_frame=3, n_freq=513, ch=1) and dtype is float
model.add(kapre.ConcatenateFrequencyMap())
# (batch, n_frame=3, n_freq=513, ch=2)
# now add your model
mode.add(keras.layers.Conv2D(16, (3, 3), strides=(2, 2), activation='relu')
# you can concatenate frequency map before other conv layers,
# but probably, you wouldn't want to add it right before batch normalization.
model.add(kapre.ConcatenateFrequencyMap())
model.add(keras.layers.Conv2D(32, (3, 3), strides=(1, 1), activation='relu')
model.add(keras.layers.MaxPooling2D((2, 2)))  # length of frequency axis doesn't␣
↪matter
```

References

Koutini, K., Eghbal-zadeh, H., & Widmer, G. (2019). Receptive-Field-Regularized CNN Variants for Acoustic Scene Classification. In Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019).

call(*x*)

Parameters **x** (*Tensor*) – a 2d batch (b, t, f, ch) or (b, ch, t, f)

Returns a 2d batch (b, t, f, ch + 1) or (b, ch + 1, t, f)

Return type x (*Tensor*)

6.3 signal

Signal layers.

This module includes Kapre layers that deal with audio signals (waveforms).

class kapre.signal.**Frame**(*frame_length*, *hop_length*, *pad_end=False*, *pad_value=0*, *data_format='default'*, ***kwargs*)

Frame input audio signal. It is a wrapper of *tf.signal.frame*.

Parameters

- **frame_length** (*int*) – length of a frame
- **hop_length** (*int*) – hop length aka frame rate
- **pad_end** (*bool*) – whether to pad at the end of the signal of there would be a otherwise-discarded partial frame
- **pad_value** (*int or float*) – value to use in the padding
- **data_format** (*str*) – *channels_first*, *channels_last*, or *default*
- ****kwargs** – optional keyword args for *tf.keras.layers.Layer()*

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.Frame(frame_length=1024, hop_length=512, input_shape=input_shape))
# now the shape is (batch, n_frame=3, frame_length=1024, ch=1)
```

call(*x*)

Parameters **x** (*Tensor*) – batch audio signal in the specified 1D format in initiation.

Returns A framed tensor. The shape is (batch, time (frames), frame_length, channel) if *channels_last*, or (batch, channel, time (frames), frame_length) if *channels_first*.

Return type (*Tensor*)

class kapre.signal.**Energy**(*sample_rate=22050*, *ref_duration=0.1*, *frame_length=2205*, *hop_length=1102*, *pad_end=False*, *pad_value=0*, *data_format='default'*, ***kwargs*)

Compute energy of each frame. The energy computed for each frame then is normalized so that the values would represent energy per *ref_duration*. I.e., if *frame_length* > *sample_rate* * *ref_duration*,

Parameters

- **sample_rate** (*int*) – sample rate of the audio
- **ref_duration** (*float*) – reference duration for normalization
- **frame_length** (*int*) – length of a frame that is used in computing energy
- **hop_length** (*int*) – hop length aka frame rate. time resolution of the energy computation.
- **pad_end** (*bool*) – whether to pad at the end of the signal of there would be a otherwise-discarded partial frame
- **pad_value** (*int or float*) – value to use in the padding

- **data_format** (*str*) – *channels_first*, *channels_last*, or *default*
- ****kwargs** – optional keyword args for *tf.keras.layers.Layer()*

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.Energy(frame_length=1024, hop_length=512, input_shape=input_
↪shape))
# now the shape is (batch, n_frame=3, ch=1)
```

call(*x*)

Parameters **x** (*Tensor*) – batch audio signal in the specified 1D format in initiation.

Returns A framed tensor. The shape is (batch, time (frames), channel) if *channels_last*, or (batch, channel, time (frames)) if *channels_first*.

Return type (*Tensor*)

class kapre.signal.**MuLawEncoding**(*quantization_channels*, ***kwargs*)

Mu-law encoding (compression) of audio signal, in [-1, 1], to [0, quantization_channels - 1]. See Wikipedia for more details.

Parameters

- **quantization_channels** (*positive int*) – Number of channels. For 8-bit encoding, use 256.
- ****kwargs** – optional keyword args for *tf.keras.layers.Layer()*

Note: Mu-law encoding was originally developed to increase signal-to-noise ratio of signal during transmission. In deep learning, mu-law became popular by WaveNet where 8-bit (256 channels) mu-law quantization was applied to the signal so that the generation of waveform amplitudes became a single-label 256-class classification problem.

Example

```
input_shape = (2048, 1)  # mono signal (float in [-1, 1])
model = Sequential()
model.add(kapre.MuLawEncoding(quantization_channels=256, input_shape=input_shape))
# now the shape is (batch, time=2048, ch=1) with int in [0, quantization_channels␣
↪- 1]
```

call(*x*)

Parameters **x** (float *Tensor*) – audio signal to encode. Shape doesn't matter.

Returns mu-law encoded x. Shape doesn't change.

Return type (int *Tensor*)

class kapre.signal.**MuLawDecoding**(*quantization_channels*, ***kwargs*)

Mu-law decoding (expansion) of mu-law encoded audio signal to [-1, 1]. See Wikipedia for more details.

Parameters

- **quantization_channels** (*positive int*) – Number of channels. For 8-bit encoding, use 256.
- ****kwargs** – optional keyword args for *tf.keras.layers.Layer()*

Example

```
input_shape = (2048, 1)  # mono signal (int in [0, quantization_channels - 1])
model = Sequential()
model.add(kapre.MuLawDecoding(quantization_channels=256, input_shape=input_shape))
# now the shape is (batch, time=2048, ch=1) with float dtype in [-1, 1]
```

call(*x*)

Parameters **x** (int *Tensor*) – audio signal to decode. Shape doesn't matter.

Returns mu-law encoded x. Shape doesn't change.

Return type (float *Tensor*)

class kapre.signal.**LogmelToMFCC**(*n_mfccs=20*, *data_format='default'*, ***kwargs*)

Compute MFCC from log-melspectrogram.

It wraps *tf.signal.mfccs_from_log_mel_spectrogram()*, which performs DCT-II.

Note: In librosa, the DCT-II scales by *sqrt(1/n)* where *n* is the bin index of MFCC as it uses scipy. This is the correct orthogonal DCT. In Tensorflow though, because it follows HTK, it scales by *(0.5 * sqrt(2/n))*. This results in *sqrt(2)* scale difference in the first MFCC bins (*n=1*).

As long as all of your data in training / inference / deployment is consistent (i.e., do not mix librosa and kapre MFCC), it'll be fine!

Parameters

- **n_mfccs** (*int*) – Number of MFCC
- **data_format** (*str*) – *channels_first*, *channels_last*, or *default*
- ****kwargs** – optional keyword args for *tf.keras.layers.Layer()*

Example

```
input_shape = (40, 128, 1)  # mono melspectrogram with 40 frames and n_mels=128
model = Sequential()
model.add(kapre.LogmelToMFCC(n_mfccs=20, input_shape=input_shape))
# now the shape is (batch, time=40, n_mfccs=20, ch=1)
```

call(*log_melgrams*)

Parameters **log_melgrams** (float *Tensor*) – a batch of log_melgrams. *(b, time, mel, ch)* if *channels_last* and *(b, ch, time, mel)* if *channels_first*.

Returns MFCCs. *(batch, time, n_mfccs, ch)* if *channels_last*, *(batch, ch, time, n_mfccs)* if *channels_first*.

Return type (float *Tensor*)

6.4 composed

Functions that returns high-level layers that are composed using other Kapre layers.

Warning: Functions in this module returns a composed Keras layer, which is an instance of *keras.Sequential* or *keras.Functional*. They are not compatible with *keras.load_model()* if *save_format == 'h5'*. The solution would be to use *save_format='tf'* (*.pb* file format). Or, you can decompose the returned layers and add it to your model manually. E.g.,

```
your_model = keras.Sequentual()
composed_melgram_layer = kapre.composed.get_melspectrogram_layer(input_shape=(44100,
↪ 1))
for layer in composed_melgram_layer.layers:
    your_model.add(layer)
```

kapre.composed.**get_stft_magnitude_layer**(*input_shape=None*, *n_fft=2048*, *win_length=None*, *hop_length=None*, *window_name=None*, *pad_begin=False*, *pad_end=False*, *return_decibel=False*, *db_amin=1e-05*, *db_ref_value=1.0*, *db_dynamic_range=80.0*, *input_data_format='default'*, *output_data_format='default'*, *name='stft_magnitude'*)

A function that returns a stft magnitude layer. The layer is a *keras.Sequential* model consists of *STFT*, *Magnitude*, and optionally *MagnitudeToDecibel*.

Parameters

- **input_shape** (*None or tuple of integers*) – input shape of the model. Necessary only if this melspectrogram layer is is the first layer of your model (see *keras.model.Sequential()* for more details)
- **n_fft** (*int*) – number of FFT points in *STFT*
- **win_length** (*int*) – window length of *STFT*
- **hop_length** (*int*) – hop length of *STFT*
- **window_name** (*str or None*) – *Name* of *tf.signal* function that returns a 1D tensor window that is used in analysis. Defaults to *hann_window* which uses *tf.signal.hann_window*. Window availability depends on Tensorflow version. More details are at *kapre.backend.get_window()*.
- **pad_begin** (*bool*) – Whether to pad with zeros along time axis (length: win_length - hop_length). Defaults to *False*.
- **pad_end** (*bool*) – whether to pad the input signal at the end in *STFT*.
- **return_decibel** (*bool*) – whether to apply decibel scaling at the end
- **db_amin** (*float*) – noise floor of decibel scaling input. See *MagnitudeToDecibel* for more details.
- **db_ref_value** (*float*) – reference value of decibel scaling. See *MagnitudeToDecibel* for more details.
- **db_dynamic_range** (*float*) – dynamic range of the decibel scaling result.

- **input_data_format** (*str*) – the audio data format of input waveform batch. *'channels_last'* if it's *(batch, time, channels)* *'channels_first'* if it's *(batch, channels, time)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **output_data_format** (*str*) – the data format of output melspectrogram. *'channels_last'* if you want *(batch, time, frequency, channels)* *'channels_first'* if you want *(batch, channels, time, frequency)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **name** (*str*) – name of the returned layer

Note: STFT magnitude represents a linear-frequency spectrum of audio signal and probably the most popular choice when it comes to audio analysis in general. By using magnitude, this layer discard the phase information, which is generally known to be irrelevant to human auditory perception.

Note: For audio analysis (when the output is tag/label/etc), we'd like to recommend to set *return_decibel=True*. Decibel scaling is perceptually plausible and numerically stable (related paper: A Comparison of Audio Signal Preprocessing Methods for Deep Neural Networks on Music Tagging) Many music, speech, and audio applications have used this log-magnitude STFT, e.g., Learning to Pinpoint Singing Voice from Weakly Labeled Examples, Joint Beat and Downbeat Tracking with Recurrent Neural Networks, and many more.

For audio processing (when the output is audio signal), it might be better to use STFT as it is (*return_decibel=False*). Example: Singing voice separation with deep U-Net convolutional networks. This is because decibel scaling is has some clipping at the noise floor which is irreversible. One may use *log(1+X)* instead of *log(X)* to avoid the clipping but it is not included in Kapre at the moment.

Example

```
input_shape = (2048, 1)  # mono signal, audio is channels_last
stft_mag = get_stft_magnitude_layer(input_shape=input_shape, n_fft=1024, return_
↪decibel=True,
    input_data_format='channels_last', output_data_format='channels_first')
model = Sequential()
model.add(stft_mag)
# now the shape is (batch, ch=1, n_frame=3, n_freq=513) because output_data_
↪format is 'channels_first'
# and the dtype is float
```

kapre.composed.**get_melspectrogram_layer**(*input_shape=None*, *n_fft=2048*, *win_length=None*, *hop_length=None*, *window_name=None*, *pad_begin=False*, *pad_end=False*, *sample_rate=22050*, *n_mels=128*, *mel_f_min=0.0*, *mel_f_max=None*, *mel_htk=False*, *mel_norm='slaney'*, *return_decibel=False*, *db_amin=1e-05*, *db_ref_value=1.0*, *db_dynamic_range=80.0*, *input_data_format='default'*, *output_data_format='default'*, *name='melspectrogram'*)

A function that returns a melspectrogram layer, which is a *keras.Sequential* model consists of *STFT*, *Magnitude*, *ApplyFilterbank(_mel_filterbank)*, and optionally *MagnitudeToDecibel*.

Parameters

- **input_shape** (*None or tuple of integers*) – input shape of the model. Necessary only if this melspectrogram layer is is the first layer of your model (see *keras.model.Sequential()* for more details)
- **n_fft** (*int*) – number of FFT points in *STFT*
- **win_length** (*int*) – window length of *STFT*
- **hop_length** (*int*) – hop length of *STFT*
- **window_name** (*str or None*) – *Name* of *tf.signal* function that returns a 1D tensor window that is used in analysis. Defaults to *hann_window* which uses *tf.signal.hann_window*. Window availability depends on Tensorflow version. More details are at *kapre.backend.get_window()*.
- **pad_begin** (*bool*) – Whether to pad with zeros along time axis (length: win_length - hop_length). Defaults to *False*.
- **pad_end** (*bool*) – whether to pad the input signal at the end in *STFT*.
- **sample_rate** (*int*) – sample rate of the input audio
- **n_mels** (*int*) – number of mel bins in the mel filterbank
- **mel_f_min** (*float*) – lowest frequency of the mel filterbank
- **mel_f_max** (*float*) – highest frequency of the mel filterbank
- **mel_htk** (*bool*) – whether to follow the htk mel filterbank fomula or not
- **mel_norm** (*'slaney' or int*) – normalization policy of the mel filterbank triangles
- **return_decibel** (*bool*) – whether to apply decibel scaling at the end
- **db_amin** (*float*) – noise floor of decibel scaling input. See *MagnitudeToDecibel* for more details.
- **db_ref_value** (*float*) – reference value of decibel scaling. See *MagnitudeToDecibel* for more details.
- **db_dynamic_range** (*float*) – dynamic range of the decibel scaling result.
- **input_data_format** (*str*) – the audio data format of input waveform batch. *'channels_last'* if it's *(batch, time, channels) 'channels_first'* if it's *(batch, channels, time)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **output_data_format** (*str*) – the data format of output melspectrogram. *'channels_last'* if you want *(batch, time, frequency, channels) 'channels_first'* if you want *(batch, channels, time, frequency)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **name** (*str*) – name of the returned layer

Note: Melspectrogram is originally developed for speech applications and has been *very* widely used for audio signal analysis including music information retrieval. As its mel-axis is a non-linear compression of (linear) frequency axis, a melspectrogram can be an efficient choice as an input of a machine learning model. We recommend to set *return_decibel=True*.

References: Automatic tagging using deep convolutional neural networks, Deep content-based music recommendation, CNN Architectures for Large-Scale Audio Classification, Multi-label vs. combined single-label sound event detection with deep neural networks, Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification, and way too many speech applications.

Example

```
input_shape = (2, 2048)  # stereo signal, audio is channels_first
melgram = get_melspectrogram_layer(input_shape=input_shape, n_fft=1024, return_
↪decibel=True,
    n_mels=96, input_data_format='channels_first', output_data_format='channels_
↪last')
model = Sequential()
model.add(melgram)
# now the shape is (batch, n_frame=3, n_mels=96, n_ch=2) because output_data_
↪format is 'channels_last'
# and the dtype is float
```

kapre.composed.**get_log_frequency_spectrogram_layer**(*input_shape=None*, *n_fft=2048*, *win_length=None*, *hop_length=None*, *window_name=None*, *pad_begin=False*, *pad_end=False*, *sample_rate=22050*, *log_n_bins=84*, *log_f_min=None*, *log_bins_per_octave=12*, *log_spread=0.125*, *return_decibel=False*, *db_amin=1e-05*, *db_ref_value=1.0*, *db_dynamic_range=80.0*, *input_data_format='default'*, *output_data_format='default'*, *name='log_frequency_spectrogram'*)

A function that returns a log-frequency STFT layer, which is a *keras.Sequential* model consists of *STFT*, *Magnitude*, *ApplyFilterbank(_log_filterbank)*, and optionally *MagnitudeToDecibel*.

Parameters

- **input_shape** (*None or tuple of integers*) – input shape of the model if this melspectrogram layer is is the first layer of your model (see *keras.model.Sequential()* for more details)
- **n_fft** (*int*) – number of FFT points in *STFT*
- **win_length** (*int*) – window length of *STFT*
- **hop_length** (*int*) – hop length of *STFT*
- **window_name** (*str or None*) – *Name* of *tf.signal* function that returns a 1D tensor window that is used in analysis. Defaults to *hann_window* which uses *tf.signal.hann_window*. Window availability depends on Tensorflow version. More details are at *kapre.backend.get_window()*.
- **pad_begin** (*bool*) – Whether to pad with zeros along time axis (length: win_length - hop_length). Defaults to *False*.
- **pad_end** (*bool*) – whether to pad the input signal at the end in *STFT*.
- **sample_rate** (*int*) – sample rate of the input audio
- **log_n_bins** (*int*) – number of the bins in the log-frequency filterbank
- **log_f_min** (*float*) – lowest frequency of the filterbank
- **log_bins_per_octave** (*int*) – number of bins in each octave in the filterbank

- **log_spread** (*float*) – spread constant (Q value) in the log filterbank.
- **return_decibel** (*bool*) – whether to apply decibel scaling at the end
- **db_amin** (*float*) – noise floor of decibel scaling input. See *MagnitudeToDecibel* for more details.
- **db_ref_value** (*float*) – reference value of decibel scaling. See *MagnitudeToDecibel* for more details.
- **db_dynamic_range** (*float*) – dynamic range of the decibel scaling result.
- **input_data_format** (*str*) – the audio data format of input waveform batch. *'channels_last'* if it's *(batch, time, channels) 'channels_first'* if it's *(batch, channels, time)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **output_data_format** (*str*) – the data format of output mel spectrogram. *'channels_last'* if you want *(batch, time, frequency, channels) 'channels_first'* if you want *(batch, channels, time, frequency)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **name** (*str*) – name of the returned layer

Note: Log-frequency spectrogram is similar to melspectrogram but its frequency axis is perfectly linear to octave scale. For some pitch-related applications, a log-frequency spectrogram can be a good choice.

Example

```
input_shape = (2048, 2)  # stereo signal, audio is channels_last
logfreq_stft_mag = get_log_frequency_spectrogram_layer(
    input_shape=input_shape, n_fft=1024, return_decibel=True,
    log_n_bins=84, input_data_format='channels_last', output_data_format=
↪'channels_last')
model = Sequential()
model.add(logfreq_stft_mag)
# now the shape is (batch, n_frame=3, n_bins=84, n_ch=2) because output_data_
↪format is 'channels_last'
# and the dtype is float
```

kapre.composed.**get_perfectly_reconstructing_stft_istft**(*stft_input_shape=None*, *istft_input_shape=None*, *n_fft=2048*, *win_length=None*, *hop_length=None*, *forward_window_name=None*, *waveform_data_format='default'*, *stft_data_format='default'*, *stft_name='stft'*, *istft_name='istft'*)

A function that returns two layers, stft and inverse stft, which would be perfectly reconstructing pair.

Parameters

- **stft_input_shape** (*tuple*) – Input shape of single waveform. Must specify this if the returned stft layer is going to be used as first layer of a Sequential model.

- **istft_input_shape** (*tuple*) – Input shape of single STFT. Must specify this if the returned istft layer is going to be used as first layer of a Sequential model.
- **n_fft** (*int*) – Number of FFTs. Defaults to *2048*
- **win_length** (*int* or *None*) – Window length in sample. Defaults to *n_fft*.
- **hop_length** (*int* or *None*) – Hop length in sample between analysis windows. Defaults to *n_fft // 4* following librosa.
- **forward_window_name** (function or *None*) – *Name* of *tf.signal* function that returns a 1D tensor window that is used. Defaults to *hann_window* which uses *tf.signal.hann_window*. Window availability depends on Tensorflow version. More details are at *kapre.backend.get_window()*.
- **waveform_data_format** (*str*) – The audio data format of waveform batch. *'channels_last'* if it's *(batch, time, channels)* *'channels_first'* if it's *(batch, channels, time)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **stft_data_format** (*str*) – The data format of STFT. *'channels_last'* if you want *(batch, time, frequency, channels)* *'channels_first'* if you want *(batch, channels, time, frequency)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **stft_name** (*str*) – name of the returned STFT layer
- **istft_name** (*str*) – name of the returned ISTFT layer

Note: Without a careful setting, *tf.signal.stft* and *tf.signal.istft* is not perfectly reconstructing.

Note: Imagine *x –> STFT –> InverseSTFT –> y*. The length of *x* will be longer than *y* due to the padding at the beginning and the end. To compare them, you would need to trim *y* along time axis.

The formula: if *trim_begin = win_length - hop_length* and *len_signal* is length of *x*, *y_trimmed = y[trim_begin: trim_begin + len_signal, :]* (in the case of *channels_last*).

Example

```
stft_input_shape = (2048, 2)  # stereo and channels_last
stft_layer, istft_layer = get_perfectly_reconstructing_stft_istft(
    stft_input_shape=stft_input_shape
)

unet = get_unet()  input: stft (complex value), output: stft (complex value)

model = Sequential()
model.add(stft_layer)  # input is waveform
model.add(unet)
model.add(istft_layer)  # output is also waveform
```

kapre.composed.**get_stft_mag_phase**(*input_shape*, *n_fft=2048*, *win_length=None*, *hop_length=None*, *window_name=None*, *pad_begin=False*, *pad_end=False*, *return_decibel=False*, *db_amin=1e-05*, *db_ref_value=1.0*, *db_dynamic_range=80.0*, *input_data_format='default'*, *output_data_format='default'*, *name='stft_mag_phase'*)

A function that returns magnitude and phase of input audio.

Parameters

- **input_shape** (*None or tuple of integers*) – input shape of the stft layer. Because this mag_phase is based on keras.Functional model, it is required to specify the input shape. E.g., (44100, 2) for 44100-sample stereo audio with *input_data_format=='channels_last'*.
- **n_fft** (*int*) – number of FFT points in *STFT*
- **win_length** (*int*) – window length of *STFT*
- **hop_length** (*int*) – hop length of *STFT*
- **window_name** (*str or None*) – *Name* of *tf.signal* function that returns a 1D tensor window that is used in analysis. Defaults to *hann_window* which uses *tf.signal.hann_window*. Window availability depends on Tensorflow version. More details are at *kapre.backend.get_window()*
- **pad_begin** (*bool*) – Whether to pad with zeros along time axis (length: win_length - hop_length). Defaults to *False*.
- **pad_end** (*bool*) – whether to pad the input signal at the end in *STFT*.
- **return_decibel** (*bool*) – whether to apply decibel scaling at the end
- **db_amin** (*float*) – noise floor of decibel scaling input. See *MagnitudeToDecibel* for more details.
- **db_ref_value** (*float*) – reference value of decibel scaling. See *MagnitudeToDecibel* for more details.
- **db_dynamic_range** (*float*) – dynamic range of the decibel scaling result.
- **input_data_format** (*str*) – the audio data format of input waveform batch. *'channels_last'* if it's *(batch, time, channels) 'channels_first'* if it's *(batch, channels, time)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **output_data_format** (*str*) – the data format of output mel spectrogram. *'channels_last'* if you want *(batch, time, frequency, channels) 'channels_first'* if you want *(batch, channels, time, frequency)* Defaults to the setting of your Keras configuration. (tf.keras.backend.image_data_format())
- **name** (*str*) – name of the returned layer

Example

```
input_shape = (2048, 3)  # stereo and channels_last
model = Sequential()
model.add(
    get_stft_mag_phase(input_shape=input_shape, return_decibel=True, n_fft=1024)
)
# now output shape is (batch, n_frame=3, freq=513, ch=6). 6 channels = [3 mag ch;␣
↪3 phase ch]
```

kapre.composed.**get_frequency_aware_conv2d**(*data_format='default'*, *freq_aware_name='frequency_aware_conv2d'*, **args*, ***kwargs*)

Returns a frequency-aware conv2d layer.

Parameters

- **data_format** (*str*) – specifies the data format of batch input/output.
- **freq_aware_name** (*str*) – name of the returned layer
- ***args** – position args for *keras.layers.Conv2D*.
- ****kwargs** – keyword args for *keras.layers.Conv2D*.

Returns A sequential model of ConcatenateFrequencyMap and Conv2D.

References

Koutini, K., Eghbal-zadeh, H., & Widmer, G. (2019). Receptive-Field-Regularized CNN Variants for Acoustic Scene Classification. In Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019).

6.5 backend

Backend operations of Kapre.

This module summarizes operations and functions that are used in Kapre layers.

kapre.backend.**_CH_FIRST_STR**
'channels_first', a pre-defined string.

Type str

kapre.backend.**_CH_LAST_STR**
'channels_last', a pre-defined string.

Type str

kapre.backend.**_CH_DEFAULT_STR**
'default', a pre-defined string.

Type str

kapre.backend.**get_window_fn**(*window_name=None*)
Return a window function given its name. This function is used inside layers such as *STFT* to get a window function.

Parameters

- **window_name** (*None or str*) – name of window function. On Tensorflow 2.3, there are five windows available in
- **tf.signal** (*hamming_window*, *hann_window*, *kaiser_bessel_derived_window*, *kaiser_window*, *vorbis_window*) –

kapre.backend.**validate_data_format_str**(*data_format*)
A function that validates the data format string.

kapre.backend.**magnitude_to_decibel**(*x*, *ref_value=1.0*, *amin=1e-05*, *dynamic_range=80.0*)
A function that converts magnitude to decibel scaling. In essence, it runs *10 * log10(x)*, but with some other utility operations.

Similar to *librosa.power_to_db* with *ref=1.0* and *top_db=dynamic_range*

Parameters

- **x** (*Tensor*) – float tensor. Can be batch or not. Something like magnitude of STFT.

- **ref_value** (*float*) – an input value that would become 0 dB in the result. For spectrogram magnitudes, ref_value=1.0 usually make the decibel-scaled output to be around zero if the input audio was in [-1, 1].
- **amin** (*float*) – the noise floor of the input. An input that is smaller than *amin*, it's converted to *amin*.
- **dynamic_range** (*float*) – range of the resulting value. E.g., if the maximum magnitude is 30 dB, the noise floor of the output would become (30 - dynamic_range) dB

Returns a decibel-scaled version of *x*.

Return type log_spec (*Tensor*)

Note: In many deep learning based application, the input spectrogram magnitudes (e.g., abs(STFT)) are decibel-scaled (=logarithmically mapped) for a better performance.

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.Frame(frame_length=1024, hop_length=512, input_shape=input_shape))
# now the shape is (batch, n_frame=3, frame_length=1024, ch=1)
```

kapre.backend.**filterbank_mel**(*sample_rate*, *n_freq*, *n_mels=128*, *f_min=0.0*, *f_max=None*, *htk=False*, *norm='slaney'*)

A wrapper for librosa.filters.mel that additionally does transpose and tensor conversion

Parameters

- **sample_rate** (*int*) – sample rate of the input audio
- **n_freq** (*int*) – number of frequency bins in the input STFT magnitude.
- **n_mels** (*int*) – the number of mel bands
- **f_min** (*float*) – lowest frequency that is going to be included in the mel filterbank (Hertz)
- **f_max** (*float*) – highest frequency that is going to be included in the mel filterbank (Hertz)
- **htk** (*bool*) – whether to use *htk* formula or not
- **norm** – The default, 'slaney', would normalize the the mel weights by the width of the mel band.

Returns mel filterbanks. Shape='(n_freq, n_mels)'

Return type (*Tensor*)

kapre.backend.**filterbank_log**(*sample_rate*, *n_freq*, *n_bins=84*, *bins_per_octave=12*, *f_min=None*, *spread=0.125*)

A function that returns a approximation of constant-Q filter banks for a fixed-window STFT. Each filter is a log-normal window centered at the corresponding frequency.

Parameters

- **sample_rate** (*int*) – audio sampling rate
- **n_freq** (*int*) – number of the input frequency bins. E.g., *n_fft / 2 + 1*
- **n_bins** (*int*) – number of the resulting log-frequency bins. Defaults to 84 (7 octaves).

- **bins_per_octave** (*int*) – number of bins per octave. Defaults to 12 (semitones).
- **f_min** (*float*) – lowest frequency that is going to be included in the log filterbank. Defaults to *C1 ~= 32.70*
- **spread** (*float*) – spread of each filter, as a fraction of a bin.

Returns log-frequency filterbanks. Shape='(n_freq, n_bins)'

Return type (*Tensor*)

Note: The code is originally from *logfrequency* in librosa 0.4 (deprecated) and copy-and-pasted. *tuning* parameter was removed and we use *n_freq* instead of *n_fft*.

---

kapre.backend.**mu_law_encoding**(*signal*, *quantization_channels*)

Encode signal based on mu-law companding. Also called mu-law compressing.

This algorithm assumes the signal has been scaled to between -1 and 1 and returns a signal encoded with values from 0 to quantization_channels - 1. See Wikipedia for more details.

Parameters

- **signal** (float *Tensor*) – audio signal to encode
- **quantization_channels** (*positive int*) – Number of channels. For 8-bit encoding, use 256.

Returns mu-encoded signal

Return type signal_mu (int *Tensor*)

kapre.backend.**mu_law_decoding**(*signal_mu*, *quantization_channels*)

Decode mu-law encoded signals based on mu-law companding. Also called mu-law expanding.

See Wikipedia for more details.

Parameters

- **signal_mu** (int *Tensor*) – mu-encoded signal to decode
- **quantization_channels** (*positive int*) – Number of channels. For 8-bit encoding, use 256.

Returns decoded audio signal

Return type signal (float *Tensor*)

6.6 time_frequency_tflite

Tflite compatible versions of Kapre layers.

STFTTflite is a tflite compatible version of *STFT*. Tflite does not support complex types, thus real and imaginary parts are returned as an extra (last) dimension. Ouput shape is now: *(batch, channel, time, re/im)* or *(batch, time, channel, re/im)*.

Because of the change of dimension, Tflite compatible layers are provided to process the resulting STFT; *MagnitudeTflite* and *PhaseTflite* are layers that calculate the magnitude and phase respectively from the output of *STFTTflite*.

class kapre.time_frequency_tflite.**STFTTflite**(*n_fft=2048*, *win_length=None*, *hop_length=None*, *window_name=None*, *pad_begin=False*, *pad_end=False*, *input_data_format='default'*, *output_data_format='default'*, ***kwargs*)

A Short-time Fourier transform layer (tflite compatible).

Ues *stft_tflite* from tflite_compatible_stft.py, this contains a tflite compatible stft (using a rdft), and *fixed_frame()* to window the audio. Tflite does not cope with comple types so real and imaginary parts are stored in extra dim. Ouput shape is now: (batch, channel, time, re/im) or (batch, time, channel, re/im). *MagnitudeTflite*, and *PhaseTflite* are versions of the *Magnitude* and *Phase* layers that account for this extra dimensionality. Currently this layer is restricted to a batch size of one, for training use the *STFT* layer, and once complete transfer the weights to a new model, replacing the *STFT* layer with the *STFTTflite* layer and *Magnitude* and *Phase* layers with *MagnitudeTflite* and *PhaseTflite* layers.

Additionally, it reshapes the output to be a proper 2D batch.

If *output_data_format == 'channels_last'*, the output shape is *(batch, time, freq, channel, re/imag)* If *output_data_format == 'channels_first'*, the output shape is *(batch, channel, time, freq, re/imag)*

Parameters

- **n_fft** (*int*) – Number of FFTs. Defaults to *2048*
- **win_length** (*int or None*) – Window length in sample. Defaults to *n_fft*.
- **hop_length** (*int or None*) – Hop length in sample between analysis windows. Defaults to *n_fft // 4* following Librosa.
- **window_name** (*str or None*) – *Name* of *tf.signal* function that returns a 1D tensor window that is used in analysis. Defaults to *hann_window* which uses *tf.signal.hann_window*. Window availability depends on Tensorflow version. More details are at *kapre.backend.get_window()*.
- **pad_begin** (*bool*) – Whether to pad with zeros along time axis (length: win_length - hop_length). Defaults to *False*.
- **pad_end** (*bool*) – Whether to pad with zeros at the finishing end of the signal.
- **input_data_format** (*str*) – the audio data format of input waveform batch. *'channels_last'* if it's *(batch, time, channels)* and *'channels_first'* if it's *(batch, channels, time)*. Defaults to the setting of your Keras configuration. (*tf.keras.backend.image_data_format()*)
- **output_data_format** (*str*) – The data format of output STFT. *'channels_last'* if you want *(batch, time, frequency, channels)* and *'channels_first'* if you want *(batch, channels, time, frequency)* Defaults to the setting of your Keras configuration. (*tf.keras.backend.image_data_format()*)
- ****kwargs** – Keyword args for the parent keras layer (e.g., *name*)

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()  # tflite compatible model
model.add(kapre.STFTTflite(n_fft=1024, hop_length=512, input_shape=input_shape))
# now the shape is (batch, n_frame=3, n_freq=513, ch=1, re/im=2)
# and the dtype is real
```

call(*x*)

Compute STFT of the input signal. If the *time* axis is not the last axis of *x*, it should be transposed first.

Parameters **x** (float *Tensor*) – batch of audio signals, (batch, ch, time) or (batch, time, ch) based on input_data_format

Returns A STFT representation of x in a 2D batch shape. The last dimension is size two and contains the real and imaginary parts of the stft. Its shape is (batch, time, freq, ch, 2) or (batch. ch, time, freq, 2) depending on *output_data_format* and *time* is the number of frames, which is *((len_src + (win_length - hop_length) / hop_length) // win_length)* if *pad_end* is *True*. *freq* is the number of fft unique bins, which is *n_fft // 2 + 1* (the unique components of the FFT).

Return type (real *Tensor*)

class kapre.time_frequency_tflite.**MagnitudeTflite**(**args*, ***kwargs*)

Compute the magnitude of the input (tflite compatible).

The input is a real tensor, the last dimension has a size of *2* representing real and imaginary parts respectively.

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.STFTTflite(n_fft=1024, hop_length=512, input_shape=input_shape))
mode.add(MagnitudeTflite())
# now the shape is (batch, n_frame=3, n_freq=513, ch=1) and dtype is float
```

call(*x*)

Parameters **x** (real or complex *Tensor*) – input is real tensor whose last dimension has a size of *2* and represents real and imaginary parts

Returns magnitude of *x*

Return type (float *Tensor*)

class kapre.time_frequency_tflite.**PhaseTflite**(*approx_atan_accuracy=None*, ***kwargs*)

Compute the phase of the complex input in radian, resulting in a float tensor (tflite compatible).

Note TF lite does not natively support atan, used in tf.math.angle, so an approximation is provided. You may want to use this approximation if you generate data using a non-tf-lite compatible STFT (faster) but want the same approximations in the training data.

Parameters **approx_atan_accuracy** (*int*) – if *None* will use *tf.math.angle()* to calculate the phase accurately. If an *int* this is the number of iterations to calculate the approximate *atan()* using a tflite compatible method. the higher the number the more accurate e.g. *approx_atan_accuracy=29000*. You may want to experiment with adjusting this number: trading off accuracy with inference speed.

Example

```
input_shape = (2048, 1)  # mono signal
model = Sequential()
model.add(kapre.STFTTflite(n_fft=1024, hop_length=512, input_shape=input_shape))
model.add(PhaseTflite(approx_atan_accuracy=5000))
# now the shape is (batch, n_frame=3, n_freq=513, ch=1) and dtype is float
```

call(*x*)

Parameters **x** (*real*) – input is real tensor with five dimensions (last dim is re/imag)

Returns phase of *x* (Radian)

Return type (float *Tensor*)

6.7 Release Note

- 21 Jan 2022 - 0.3.7
  - Add [SpecAugment](https://github.com/keunwoochoi/kapre/pull/135) layer
- 13 Nov 2021 - 0.3.6
  - bugfix/pad end tflite #131
- 18 March 2021 - 0.3.5
  - Add *kapre.time_frequency_tflite* which uses tflite for a faster CPU inference.
- 29 Sep 2020 - 0.3.4
  - Fix a bug in *kapre.backend.get_window_fn()*. Previously, it only correctly worked with *None* input and an error was raised when non-default value was set for *window_name* in any layer.
- 15 Sep 2020 - 0.3.3
  - *kapre.augmentation* is added
  - *kapre.time_frequency.ConcatenateFrequencyMap* is added
  - *kapre.composed.get_frequency_aware_conv2d* is added
  - In *STFT* and *InverseSTFT*, keyword arg *window_fn* is renamed to *window_name* and it expects string value, not function. - With this update, models with Kapre layers can be loaded with *h5* file format.
  - *kapre.backend.get_window_fn* is added
- 28 Aug 2020 - 0.3.2
  - *kapre.signal.Frame* and *kapre.signal.Energy* are added
  - *kapre.signal.LogmelToMFCC* is added
  - *kapre.signal.MuLawEncoder* and *kapre.signal.MuLawDecoder* are added
  - *kapre.composed.get_stft_magnitude_layer()* is added
- 21 Aug 2020 - 0.3.1
  - *Inverse STFT* is added
- 15 Aug 2020 - 0.3.0
  - Breaking and simplifying changes with Tensorflow 2.0 and more tests. Some features are removed.
- 29 Jul 2020 - 0.2.0
  - Change melspectrogram filterbank from *norm=1* to *norm='slaney'* (w.r.t. Librosa) due to the update from Librosa (https://github.com/keunwoochoi/kapre/issues/77)

  This would change the behavior of melspectrogram slightly. - Bump librosa version to 0.7.2 or higher.
- 17 Mar 2020 - 0.1.8
  - added *utils.Delta* layer

- 20 Feb 2020 - Kapre ver 0.1.7
  - No vanilla Keras dependency
  - Tensorflow >= 1.15 only
  - Not tested on Python 2.7 anymore; only on Python 3.6 and 3.7 locally (by *tox*) and 3.6 on Travis
- 20 Feb 2019 - Kapre ver 0.1.4
  - Fixed amplitude-to-decibel error as raised in https://github.com/keunwoochoi/kapre/issues/46
- March 2018 - Kapre ver 0.1.3
  - Kapre is on Pip again
  - Add unit tests
  - Remove *Datasets*
  - Remove some codes while adding more dependency on Librosa to make it cleaner and more stable - and therefore *htk* option enabled in *Melspectrogram*
- 9 July 2017 - Kapre ver 0.1.1, aka 'pretty stable' with a benchmark paper, https://arxiv.org/abs/1706.05781
  - Remove STFT, python3 compatible
  - A full documentation in this readme.md
  - pip version is updated

Python Module Index

k

kapre.backend, 32
kapre.composed, 25
kapre.signal, 22
kapre.time_frequency, 16
kapre.time_frequency_tflite, 34

Index

Symbols

_CH_DEFAULT_STR (*in module kapre.backend*), 32
_CH_FIRST_STR (*in module kapre.backend*), 32
_CH_LAST_STR (*in module kapre.backend*), 32

A

ApplyFilterbank (*class in kapre.time_frequency*), 20

C

call() (*kapre.signal.Energy method*), 23
call() (*kapre.signal.Frame method*), 22
call() (*kapre.signal.LogmelToMFCC method*), 24
call() (*kapre.signal.MuLawDecoding method*), 24
call() (*kapre.signal.MuLawEncoding method*), 23
call() (*kapre.time_frequency.ApplyFilterbank method*), 20
call() (*kapre.time_frequency.ConcatenateFrequencyMap method*), 21
call() (*kapre.time_frequency.Delta method*), 20
call() (*kapre.time_frequency.InverseSTFT method*), 18
call() (*kapre.time_frequency.Magnitude method*), 18
call() (*kapre.time_frequency.MagnitudeToDecibel method*), 19
call() (*kapre.time_frequency.Phase method*), 19
call() (*kapre.time_frequency.STFT method*), 17
call() (*kapre.time_frequency_tflite.MagnitudeTflite method*), 36
call() (*kapre.time_frequency_tflite.PhaseTflite method*), 36
call() (*kapre.time_frequency_tflite.STFTTflite method*), 35
ConcatenateFrequencyMap (*class in kapre.time_frequency*), 21

D

Delta (*class in kapre.time_frequency*), 20

E

Energy (*class in kapre.signal*), 22

F

filterbank_log() (*in module kapre.backend*), 33
filterbank_mel() (*in module kapre.backend*), 33
Frame (*class in kapre.signal*), 22

G

get_frequency_aware_conv2d() (*in module kapre.composed*), 31
get_log_frequency_spectrogram_layer() (*in module kapre.composed*), 28
get_melspectrogram_layer() (*in module kapre.composed*), 26
get_perfectly_reconstructing_stft_istft() (*in module kapre.composed*), 29
get_stft_mag_phase() (*in module kapre.composed*), 30
get_stft_magnitude_layer() (*in module kapre.composed*), 25
get_window_fn() (*in module kapre.backend*), 32

I

InverseSTFT (*class in kapre.time_frequency*), 17

K

kapre.backend (*module*), 32
kapre.composed (*module*), 25
kapre.signal (*module*), 22
kapre.time_frequency (*module*), 16
kapre.time_frequency_tflite (*module*), 34

L

LogmelToMFCC (*class in kapre.signal*), 24

M

Magnitude (*class in kapre.time_frequency*), 18
magnitude_to_decibel() (*in module kapre.backend*), 32
MagnitudeTflite (*class in kapre.time_frequency_tflite*), 36

MagnitudeToDecibel (*class in kapre.time_frequency*), 19
mu_law_decoding() (*in module kapre.backend*), 34
mu_law_encoding() (*in module kapre.backend*), 34
MuLawDecoding (*class in kapre.signal*), 23
MuLawEncoding (*class in kapre.signal*), 23

P

Phase (*class in kapre.time_frequency*), 18
PhaseTflite (*class in kapre.time_frequency_tflite*), 36

S

STFT (*class in kapre.time_frequency*), 16
STFTTflite (*class in kapre.time_frequency_tflite*), 34

V

validate_data_format_str() (*in module kapre.backend*), 32