WaveTF
A fast 2D wavelet transform for machine learning in Keras

Francesco Versaci

Visual and Data-Intensive Computing Group
CRS4, Cagliari, Italy

– CADL @ ICPR 2020 –
Motivation

- Wavelet transforms are a family of signal transformations
- They produce a mix of time/spatial and frequency data
- Countless applications, e.g., image compression, medical imaging, finance, geophysics, and astronomy
- There is a growing number of applications in machine learning
- But there were no efficient 2D wavelet libraries available for Keras
- Now there is one 😊
Main idea

Given a (even-sized) vector of real numbers we decompose it locally (i.e., by grouping few values) in

Low frequency i.e., mean values

High frequency i.e., deviation from the mean

- For example, given \( x = (x_0, \ldots, x_{n-1}) \) we define \( H(x) := (l(x), h(x)) \), where

\[
l_i := \frac{x_{2i} + x_{2i+1}}{2} \quad h_i := \frac{x_{2i} - x_{2i+1}}{2}
\]

- Given \( x = (100, 20, 40, 80, 50, 30, 50, 150) \) we have

\[
l = (60, 60, 40, 100) \quad h = (40, -20, 10, -50)
\]
The wavelet transform $H$ is often iterated on its low component, to produce a multilevel transform

$$H^d(x) := (H^{d-1}(l(x)), h(x)),$$

with $H^0(x) := x$

Example

Given $x = (100, 20, 40, 80, 50, 30, 50, 150)$ we have

$$l^1 := l(x) = (60, 60, 40, 100) \quad h^1 := h(x) = (40, -20, 10, -50)$$

and, iterating $H$ on $l^1$ and $l^2$,

$$l^2 := l(l^1) = (60, 70) \quad h^2 := h(l^1) = (0, -30)$$

$$l^3 := l(l^2) = (65) \quad h^3 := h(l^2) = (-5)$$
We can extend the wavelet to **multidimensional signals** by executing it orderly on all the dimensions.

For example, if our input is a **matrix** we first transform its **rows** and then its **columns**.

**Example**

Given \( m = \begin{pmatrix} 100 & 20 \\ 30 & 50 \end{pmatrix} \) as input, we first transform its rows:

\[
L(m) = \begin{pmatrix} 60 \\ 40 \end{pmatrix}, \quad H(m) = \begin{pmatrix} 40 \\ -10 \end{pmatrix}
\]

and finally we transform the obtained columns, obtaining:

\[
LL(m) = (50), \quad LH(m) = (10), \quad HL(m) = (15), \quad HH(m) = (25)
\]
2D Wavelet transform
Example of multilevel decomposition

- Original image vs. its wavelet transform
- Wavelet components have been contrasted to enhance their structure
2D Wavelet transform
Example of multilevel decomposition

- **Original** image vs. its **wavelet** transform
- Wavelet components have been **contrasted** to enhance their structure
2D Wavelet transform
Example of multilevel decomposition

- Original image vs. its wavelet transform
- Wavelet components have been contrasted to enhance their structure
• Original image vs. its wavelet transform
• Wavelet components have been contrasted to enhance their structure
Why WaveTF?
Available Python wavelet libraries

PyWavelets
- Most widely used Python library for wavelet transforms
- Its core routines are written in C
- Supports over 100 wavelet kernels and 9 padding modes
- Sequential library, runs exclusively on CPUs

pypwt
- Python wrapper of PDWT (C++ wavelet transform library)
- Written using the parallel CUDA platform and running on NVIDIA GPUs
- It supports 72 wavelet kernels and periodic padding

TF-Wavelets
- Written in Python for TensorFlow
- Features 2 wavelet kernels and periodic padding
- It lacks support of batched, multichannel, 2D transforms
- Does not offer Keras integration
Why WaveTF?

Desired features

- Efficient, parallel implementation, running on both CPUs and GPUs
- Easy to integrate into already existing Keras ML applications
- E.g., add wavelet layers to existing Keras CNNs
- Supports 2D, batched, multichannel inputs (i.e., input tensors of shape [batch_size, dim_x, dim_y, channels])

Types of wavelet transforms

\[ H(x) = (l(x), h(x)) \]

### Haar wavelet

\[ l_i := \frac{x_{2i} + x_{2i+1}}{\sqrt{2}} \]
\[ h_i := \frac{x_{2i} - x_{2i+1}}{\sqrt{2}} \]

### Daubechies wavelet (DB2)

\[ l_i = \lambda_0 x_{2i-1} + \lambda_1 x_{2i} + \lambda_2 x_{2i+1} + \lambda_3 x_{2i+2} \]
\[ h_i = \mu_0 x_{2i-1} + \mu_1 x_{2i} + \mu_2 x_{2i+1} + \mu_3 x_{2i+2} \]

where

\[ \lambda_0 = \frac{1+\sqrt{3}}{2\sqrt{2}} \quad \lambda_1 = \frac{3+\sqrt{3}}{2\sqrt{2}} \quad \lambda_2 = \frac{3-\sqrt{3}}{2\sqrt{2}} \quad \lambda_3 = \frac{1-\sqrt{3}}{2\sqrt{2}} \]
\[ \mu_0 = \lambda_3 \quad \mu_1 = -\lambda_2 \quad \mu_2 = \lambda_1 \quad \mu_3 = -\lambda_0 \]
Wavelet in matricial form
Daubechies DB2 direct transform

\[
\begin{pmatrix}
  l_0 & h_0 \\
  l_1 & h_1 \\
  l_2 & h_2 \\
  l_3 & h_3 \\
  \vdots & \vdots \\
  l_{\frac{n}{2}-1} & h_{\frac{n}{2}-1}
\end{pmatrix}
= \begin{pmatrix}
  2x_0 - x_1 & x_0 & x_1 & x_2 \\
  x_1 & x_2 & x_3 & x_4 \\
  x_3 & x_4 & x_5 & x_6 \\
  x_5 & x_6 & x_7 & x_8 \\
  \vdots & \vdots & \vdots & \vdots \\
  x_{n-3} & x_{n-2} & x_{n-1} & 2x_{n-1} - x_{n-2}
\end{pmatrix}
\begin{pmatrix}
  \lambda_0 & \mu_0 \\
  \lambda_1 & \mu_1 \\
  \lambda_2 & \mu_2 \\
  \lambda_3 & \mu_3
\end{pmatrix}
\]

- In general we need some **padding** to allow invertibility
- We adopt **anti-symmetric-reflect** padding, which preserves the signal’s first-order finite difference
- In **TensorFlow**, this operation can be implemented with the specialized **conv1d** method
- Or alternatively with the **reshape**, **concat** and **stack** methods
- We have tried both and adopted the fastest one when needed
Wavelet in matricial form
Daubechies DB2 inverse transform

\[
\begin{pmatrix}
    x_1 & x_2 \\
    x_3 & x_4 \\
    \vdots & \vdots \\
    x_{n-3} & x_{n-2}
\end{pmatrix}
= \begin{pmatrix}
    l_0 & h_0 & l_1 & h_1 \\
    l_1 & h_1 & l_2 & h_2 \\
    \vdots & \vdots & \vdots & \vdots \\
    l_{n-3} & h_{n-3} & l_{n-2} & h_{n-2}
\end{pmatrix}
\begin{pmatrix}
    \lambda_2 & \lambda_3 \\
    \mu_2 & \mu_3 \\
    \lambda_0 & \lambda_1 \\
    \mu_0 & \mu_1
\end{pmatrix}
\]

with border values:

\[
\begin{pmatrix}
    x_0 \\
    x_1
\end{pmatrix}
= \mathcal{W}_{00}^+ \begin{pmatrix}
    l_0 \\
    h_0 \\
    l_1 \\
    h_1
\end{pmatrix}
\]
\[
\begin{pmatrix}
    x_{n-2} \\
    x_{n-1}
\end{pmatrix}
= \mathcal{W}_{22}^+ \begin{pmatrix}
    l_{n-2} \\
    h_{n-2} \\
    l_{n-1} \\
    h_{n-1}
\end{pmatrix}
\]

- The **inverse transform** can be computed in a similar fashion as the direct one
- The details on how to reconstruct the **border values** are a bit tricky, and are spelled out at length in the paper
WaveTF
Features

- Written in Python using the TensorFlow library
- Offers a Keras layer to allow easy integration in already existing ML applications (e.g., add to CNNs)
- 2D Haar and DB2 wavelet kernels
- Supports 2D, batched, multichannel inputs (i.e., input tensors of shape [batch_size, dim_x, dim_y, channels])
- Supports both 32- and 64-bit floats transparently at runtime

Shortcomings

It currently only supports:

- Two wavelet kernels and one padding scheme
- 1D and 2D signals
The library is **free software**, under the Apache License, Version 2.0

The source code is available at [https://github.com/crs4/WaveTF](https://github.com/crs4/WaveTF)

The (very slim) documentation can be found at [https://wavetf.readthedocs.io](https://wavetf.readthedocs.io)
import tensorflow as tf
from wavetf import WaveTFFactory

# input tensor
_t0 = tf.random.uniform([32, 300, 200, 3])
# transform
_w = WaveTFFactory().build('db2', dim=2)
_t1 = _w.call(_t0)
# anti-transform
_w_i = WaveTFFactory().build('db2', dim=2, inverse=True)
_t2 = _w_i.call(_t1)
# compute difference
_delta = abs(_t2 - _t0)
print(f'Precision error: {tf.math.reduce_max(_delta)}')

> Precision error: 1.5497207641601562e-06
import tensorflow as tf
from wavetf import WaveTFFactory

# input tensor
t0 = tf.random.uniform([32, 300, 200, 3], dtype=tf.float64)
# transform
w = WaveTFFactory().build('db2', dim=2)
t1 = w.call(t0)
# anti-transform
w_i = WaveTFFactory().build('db2', dim=2, inverse=True)
t2 = w_i.call(t1)
# compute difference
delta = abs(t2-t0)
print(f'Precision error: {tf.math.reduce_max(delta)}')
Performance

We have tested WaveTF in two ways:

- on **raw signal transforms**, to assess its speed compared to the other wavelet libraries
- as a **Keras layer**, integrated in a simple neural network, to understand the **overhead** it adds to standard **ML tasks**

---

**Hardware configuration of the test machine**

<table>
<thead>
<tr>
<th><strong>CPU</strong></th>
<th>Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz (24 SMT cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RAM</strong></td>
<td>250 GiB</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td>NVIDIA GeForce RTX 2080 Ti (11 GB GDDR6)</td>
</tr>
</tbody>
</table>
Test procedure

One dimensional case:

- A **random array of \( n \) elements** is created, with \( n \) ranging from \( 5 \cdot 10^6 \) to \( 10^8 \),
- For the non-batched case the array is used as is (i.e., shape = [\( n \)], for the batched case it is reshaped to [\( b, n/b \)], with \( b = 100 \),
- The transform, on the same input array, is executed from a minimum of 500 up to a maximum of 10000 times for smaller data size
- The total time is measured and the **time per iteration** is recorded.

Two-dimensional case: The input matrix is chosen to be **as square as possible** given the target total size of \( n \) elements, i.e., shape = [\( \lfloor \sqrt{n} \rfloor, \lceil \sqrt{n} \rceil \)].
### Raw transformation

Runtimes normalized against WaveTF (using the largest tested size)

<table>
<thead>
<tr>
<th>Operation</th>
<th>WaveTF</th>
<th>TF-Wavelets</th>
<th>PyWavelets</th>
<th>pypwt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1D Haar</td>
<td>1</td>
<td>2.98</td>
<td>74.81</td>
<td>73.55</td>
</tr>
<tr>
<td>1D DB2</td>
<td>1</td>
<td>1.58</td>
<td>42.91</td>
<td>36.04</td>
</tr>
<tr>
<td>1D Haar, batched</td>
<td>1</td>
<td>3.21</td>
<td>73.69</td>
<td>72.37</td>
</tr>
<tr>
<td>1D DB2, batched</td>
<td>1</td>
<td>1.62</td>
<td>39.85</td>
<td>33.63</td>
</tr>
<tr>
<td>2D Haar</td>
<td>1</td>
<td>2.58</td>
<td>45.59</td>
<td>14.30</td>
</tr>
<tr>
<td>2D DB2</td>
<td>1</td>
<td>2.30</td>
<td>44.61</td>
<td>12.27</td>
</tr>
<tr>
<td>2D Haar, batched</td>
<td>1</td>
<td>n.a.</td>
<td>42.55</td>
<td>n.a.</td>
</tr>
<tr>
<td>2D DB2, batched</td>
<td>1</td>
<td>n.a.</td>
<td>41.08</td>
<td>n.a.</td>
</tr>
</tbody>
</table>

- The wavelet transform has **high parallelism and low computational complexity** \(O(n)\)
- To achieve good performance we need to **minimize communication** between CPU and GPU
Raw transformation
Runtimes for 2D DB2 transform

2D Daubechies-N=2

Execution time [s]

Signal size $n$

2D Daubechies-N=2

PyWavelets
WaveTF + RAM
pypwt
TF-Wavelets
WaveTF

F. Versaci (CRS4)
Keras layer in CNN
Experiment description

- We want to quantify **training and evaluation overhead** in a typical classification problem
- We adopted the **Imagenette2-320 dataset**, consisting of 9469 training and 3925 validation RGB images
- We **wavelet-enriched** a simple CNN network, featuring 5 levels of convolution followed by downscaling
Keras layer in CNN

Results

<table>
<thead>
<tr>
<th>Operation</th>
<th>Baseline</th>
<th>With wavelet</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training time [s]</td>
<td>1581 ± 18</td>
<td>1593 ± 14</td>
<td>&lt;1%</td>
</tr>
<tr>
<td>Evaluation time [s]</td>
<td>78.5 ± 0.5</td>
<td>78.7 ± 0.8</td>
<td>&lt;1%</td>
</tr>
</tbody>
</table>

- **Running times** with and without enriching the network with wavelet features computed by the WaveTF Keras layer
- We measured the wall clock time required to train the model for 20 epochs and averaged the process over 20 repetitions
- We evaluated all the images in the dataset and repeated the process 20 times
- **No data augmentation** has been performed
- The overhead is below 1%, both in training and evaluation, thus allowing its use at an almost negligible cost
Wavelet transform is a powerful tool used in many areas.

- If you want to try and integrate it in your TensorFlow/Keras applications, just download the code and start playing with it.
- It is free software and it adds negligible runtime to existing ML pipelines.
Wavelet transform is a powerful tool used in many areas

If you want to try and integrate it in your TensorFlow/Keras applications, just **download the code** and start playing with it

It is **free software** and it adds negligible runtime to existing ML pipelines

**Thanks for your attention!**