WaveTF

A fast 2D wavelet transform for machine learning in Keras

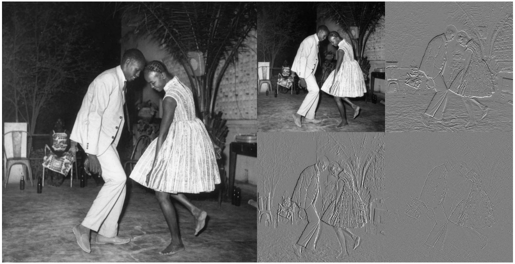
Francesco Versaci

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 😄



Original image © Malick Sidibé Wavelet transform

1D Wavelet transform

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

$$l_i := \frac{x_{2i} + x_{2i+1}}{2} \qquad h_i := \frac{x_{2i} - x_{2i+1}}{2}$$

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

$$l = (60, 60, 40, 100) \qquad h = (40, -20, 10, -50)$$

The wavelet transform H is often iterated on its low component, to produce a multilevel transform

$$H^d(x) := \left(H^{d-1}(l(x)), h(x)\right) \quad , \qquad \text{with } H^0(x) := x$$

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,

$$l^2 := l(l^1) = (60, 70) \qquad h^2 := h(l^1) = (0, -30)$$
$$l^3 := l(l^2) = (65) \qquad h^3 := h(l^2) = (-5)$$

2D Wavelet transform

- 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

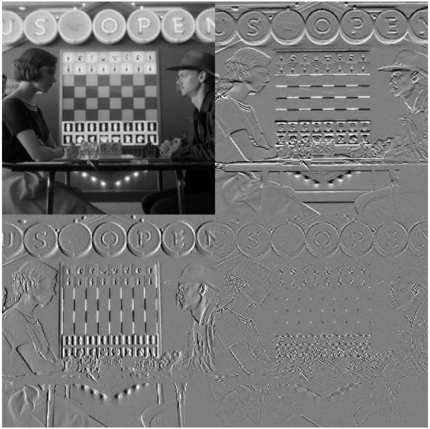
Example

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

$$L(m) = \begin{pmatrix} 60 \\ 40 \end{pmatrix} \qquad H(m) = \begin{pmatrix} 40 \\ -10 \end{pmatrix}$$

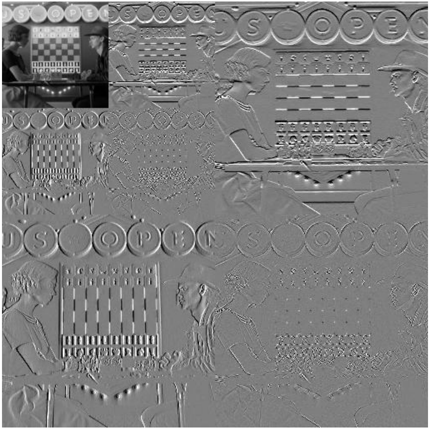
and finally we tranform the obtained columns, obtaining

$$LL(m) = (50) \quad LH(m) = (10) \quad HL(m) = (15) \quad HH(m) = (25)$$

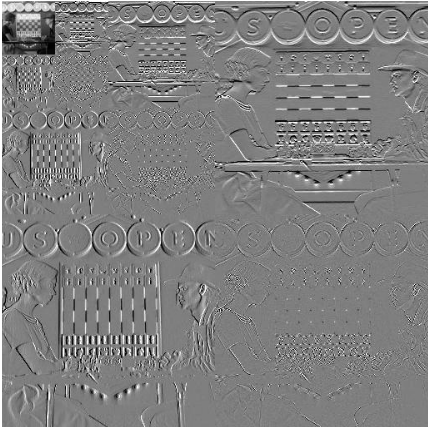


- Original image vs. its wavelet transform
- Wavelet components have been contrasted to enhance their structure

2D Wavelet transform
Example of multilevel decomposition



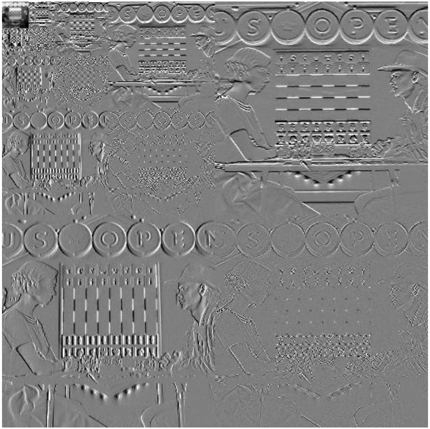
- Original image vs. its wavelet transform
- Wavelet components have been contrasted to enhance their structure



- Original image vs. its wavelet transform
- Wavelet components have been contrasted to enhance their structure

2D Wavelet transform

Example of multilevel decomposition



- Original image vs. its wavelet transform
- Wavelet components have been contrasted to enhance their structure

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

$$l_i := \frac{x_{2i} + x_{2i+1}}{\sqrt{2}} \qquad h_i := \frac{x_{2i} - x_{2i+1}}{\sqrt{2}}$$

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

$$\begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \\ \vdots & \vdots \\ x_{n-3} & x_{n-2} \end{pmatrix} = \begin{pmatrix} l_0 & h_0 & l_1 & h_1 \\ l_1 & h_1 & l_2 & h_2 \\ \vdots & \vdots & \vdots & \vdots \\ l_{\frac{n}{2}-3} & h_{\frac{n}{2}-3} & l_{\frac{n}{2}-2} & h_{\frac{n}{2}-2} \\ l_{\frac{n}{2}-2} & h_{\frac{n}{2}-2} & l_{\frac{n}{2}-1} & h_{\frac{n}{2}-1} \end{pmatrix} \begin{pmatrix} \lambda_2 & \lambda_3 \\ \mu_2 & \mu_3 \\ \lambda_0 & \lambda_1 \\ \mu_0 & \mu_1 \end{pmatrix}$$

with border values:

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = W_{00}^{+} \begin{pmatrix} l_0 \\ h_0 \\ l_1 \\ h_1 \end{pmatrix} \qquad \begin{pmatrix} x_{n-2} \\ x_{n-1} \end{pmatrix} = W_{22}^{+} \begin{pmatrix} l_{\frac{n}{2}-2} \\ h_{\frac{n}{2}-2} \\ l_{\frac{n}{2}-1} \\ h_{\frac{n}{2}-1} \end{pmatrix}$$

- 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

- 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
- The (very slim) documentation can be found at https://wavetf.readthedocs.io

```python
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
```

```python
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)}')
```

```
> Precision error:  5.329070518200751e-15
```

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

CPU	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz (24 SMT cores)
RAM	250 GiB
GPU	NVIDIA GeForce RTX 2080 Ti (11 GB GDDR6)

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

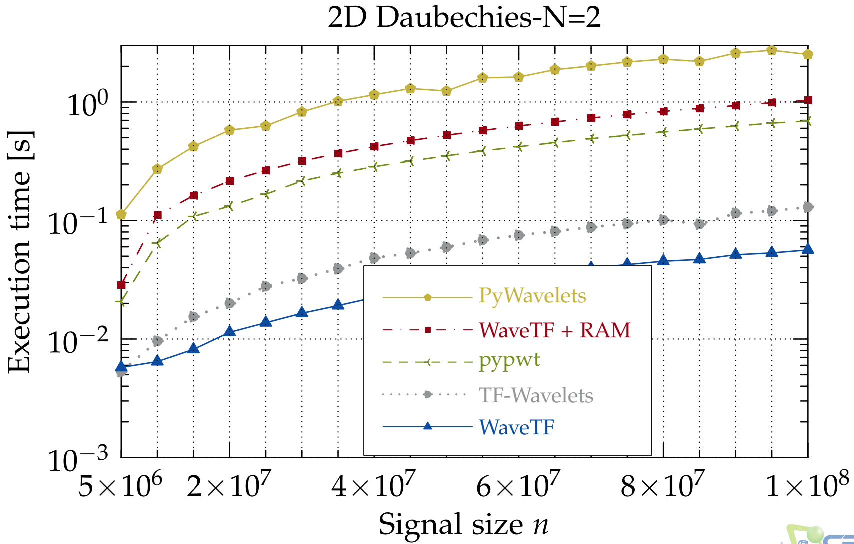
Runtimes normalized against WaveTF (using the largest tested size)

Operation	WaveTF	TF-Wavelets	PyWavelets	pypwt
1D Haar	**1**	2.98	74.81	73.55
1D DB2	**1**	1.58	42.91	36.04
1D Haar, batched	**1**	3.21	73.69	72.37
1D DB2, batched	**1**	1.62	39.85	33.63
2D Haar	**1**	2.58	45.59	14.30
2D DB2	**1**	2.30	44.61	12.27
2D Haar, batched	**1**	n.a.	42.55	n.a.
2D DB2, batched	**1**	n.a.	41.08	n.a.

- 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

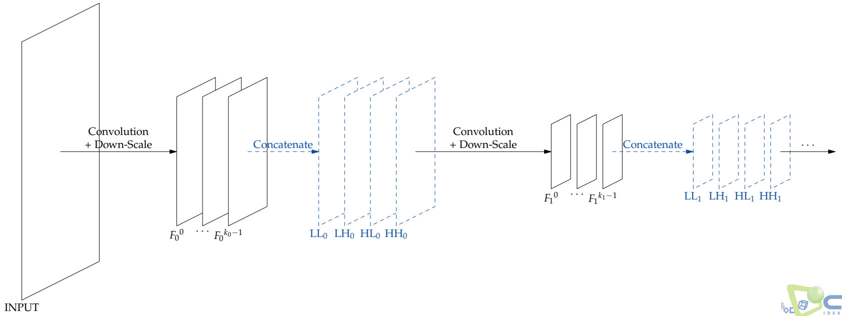
Raw transformation

Runtimes for 2D DB2 transform



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



Operation	Baseline	With wavelet	Overhead
Training time [s]	1581 ± 18	1593 ± 14	<1%
Evaluation time [s]	78.5 ± 0.5	78.7 ± 0.8	<1%

- 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

Conclusion

- 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

Conclusion

- 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!