Experimental impact of (non-differentiable) image filters on image recognition residual networks

Amélioration de la reconnaissance d'image en utilisant des filtres d'image non réversibles avec des réseaux résiduels de neurones

Marc Fiammante[1]

[1] IBM, Nice, France, www.linkedin.com/in/marc-fiammante-23074a2

RÉSUMÉ. Cet article décrit les améliorations de précision observées sur les réseaux de neurones par l'injection de filtres d'images non dérivables. En introduction il fournit aussi un petit état des lieux de techniques utilisées dans ces réseaux.

ABSTRACT. This article describes the accuracy improvements observer by the injection of mathematically non-differentiable image filters As an introduction it provides a small state of the art of the techniques used in image recognition neural networks.

MOTS-CLÉS. Reconnaissance d'image, réseaux de neurones, CNN, filtres d'images, dérivabilité, ondelettes, Keras.

KEYWORDS. Image recognition, Neural Networks, CNN, differentiability, wavelets.

1. Introduction

A while ago I was part of a project trying to locate objects on bad resolution images and even trying the most recent neural networks in the open source I was not progressing on the accuracy. Trying various approaches I added the wavelet information to the existing images and suddenly gained 12% accuracy which is significant. Then I found an approach to add other filter information to images and gained even more accuracy on this particularly case.

In this article I propose to explore some of the image recognition techniques and some information on the improvement I could get from mathematically non-differentiable filters with an example taking the public Cifar10 tiny image dataset as an example

2. Some background of image recognition

In the past years image recognition neural networks significantly evolved by using convolutions, skip connections (residual networks) and parallel layer stacks (inception). By design a neural network is a sequence of vector matrix computation with a common (mostly) differentiable function, so that the partial derivative can be computed when performing a back propagation to adjust the weights from an expected vectorized result. Image recognition networks convert all pixels of all channels into a vector that will be the input to the neural networks. By iterating on the inputs and the outputs the network will progressively learn filters that will probabilistically match a given input to an element of the output vector, elements which could be representative of classes such as dogs, cats, cars, etc.

The initially fully connected layers led to a huge number of weights and an improvement was to use convolutional layers that computed a set of convolutions identical for all pixels of an image thus reducing the number of weights while providing useful information on image features. In summary convolutions are a way to extract features from an image such as boundaries, average brightness, angles etc. that are essential for identifying shapes of objects on images.

These convolutions are the result of the computation of parameterized elements of the pixel matrix surrounding a given pixel. As example the Laplacian operator detects edges on an image by computing the second order derivative.

The mathematical formula for the Laplacian is:

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial yx^2} \quad [1]$$

In a one-dimensional world let's suppose that y1, y2 and y3 are the values of three contiguous pixels on a line and we want to compute the Laplacian at the pixel y2.

The first order derivatives are respectively

$$\text{left of pixel} = y2 - y1 \ [2]$$

$$\text{right of pixel} = y3 - y2. \ [3]$$

The second order derivative is the difference of these two which yields to 1D Laplacian:

$$1D\ Laplacian = \ (y3 - y2) - (y2 - y1) \ [4]$$

which leads to:

$$\text{1D Laplacian} = (y1 + y3 - 2 \times y2) \ [5]$$

In a discrete vector or matrix representation we get:

$$\begin{matrix} 1 & -2 & 1 \end{matrix} \ [6]$$

Extending to a two-dimensions image a commonly used discrete approximation of the Laplacian with diagonals translates to the following parameters for a convolution.

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix} \ [7]$$

Several convolutional filters are commonly used in image processing such as Sobel operator to detect boundaries, Gaussian noise reducing kernel, Laplacian of Gaussian that locates edges on images with a reduced noise.

Depending on the features the convolutions can look on more surrounding pixels and work on 5 x 5, 7 x 7 however leading to more computing power consumption.

A neural network convolutional layer will learn many different convolutions to be applied to all pixels in the image. The layers are customized with the convolution kernel sizes, 3 x 3, 5 x 5 etc. and the number of convolutional features to discover. Some layers applying pooling that reduce the dimensionality of the information by grouping information from several pixels together as an example it is not necessary to have all information from a homogeneous surface.

These networks led to significant progress in recognition, and network depth increased, simplistically stated each convolutional layer computes combinations on the previous layer features, such as an angle resulting from two lines. But the depth increase led to a loss of the initial information which is when residual networks added a skip connection that added back the initial information to the discovered features. An improvement was added with inception networks that compute in parallel convolutions with different kernel sizes and pooling that reduce dimensionality.

3. The unexpected effect of filter additions

In machine learning courses we are taught that a neural network can discover any function. That is true, but my experience makes me think this function need to be differentiable nearly everywhere.

Some of the image filters such as a variety of wavelets used by JPEG 2000 compression are not reversible, The same applies for K-Means used for making a cartoon effect from an image, Fourier lowpass filters, absolute values of image differences etc.

To help with the demonstration I reused the network from John Olafenwa's article, "Understanding Residual Networks" [OLA 18] which I like very much, and I modified the input layer to have additional filters.

To perform the addition of filters, I sub-classed a few classes from Keras such as NumpyArrayIterator so that I could compute the filters after the image augmentation provided out of the box.

The input layer then becomes a parallel separation a processing of each added filters, four filters in the example below with the number of filters from the network in the article adjusted to match the output of the new input layer

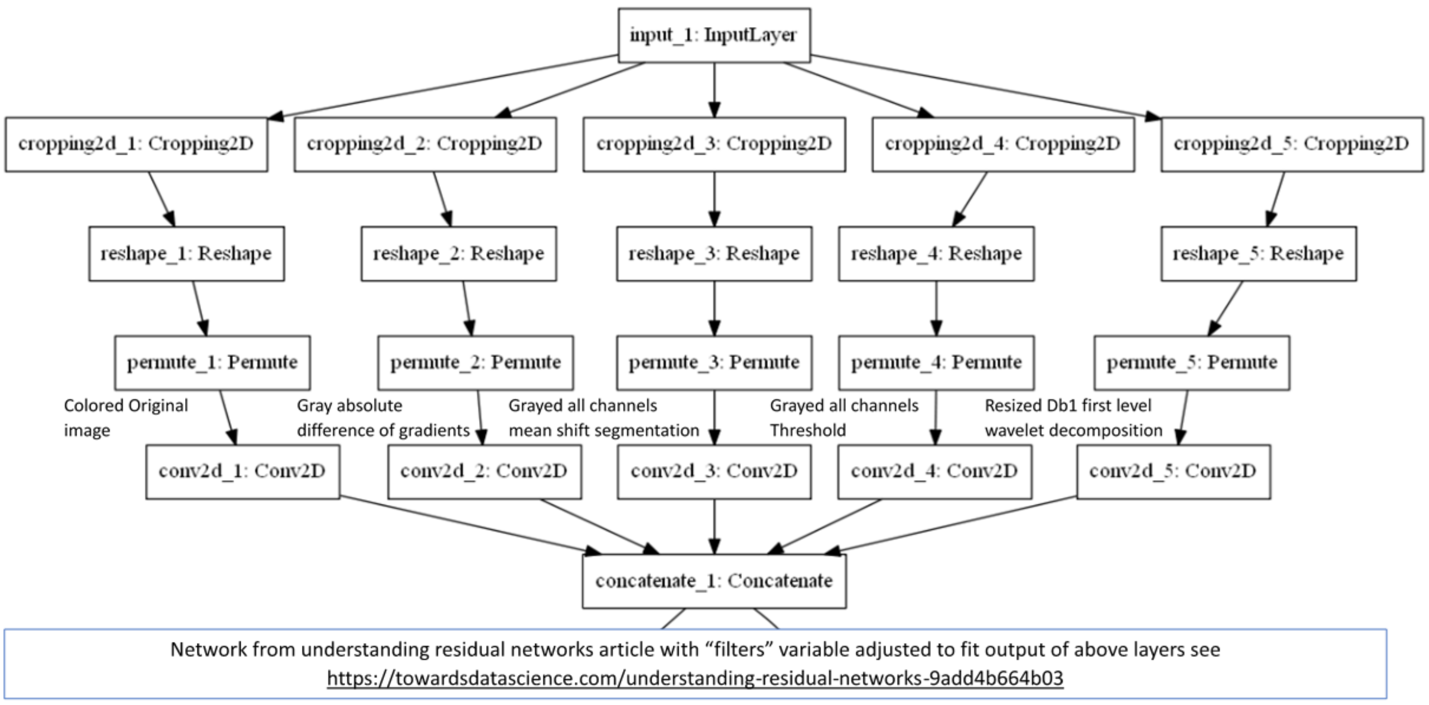

Figure 1. *Input layer addition to process filters added to the images*

The following figure shows example of possible non-reversible filters created with OpenCV image processing open source that could be used in the above enhanced input layer. As one can see from the filters some of them are quite good at revealing edges.

Another advantage of using filters is to reduce the possibility of adversarial attacks on neural networks such as with image poisoning. The low pass or smoothing filters will remove any pixel size poisons, and the addition of filters in parallel towers given them a high input weight.

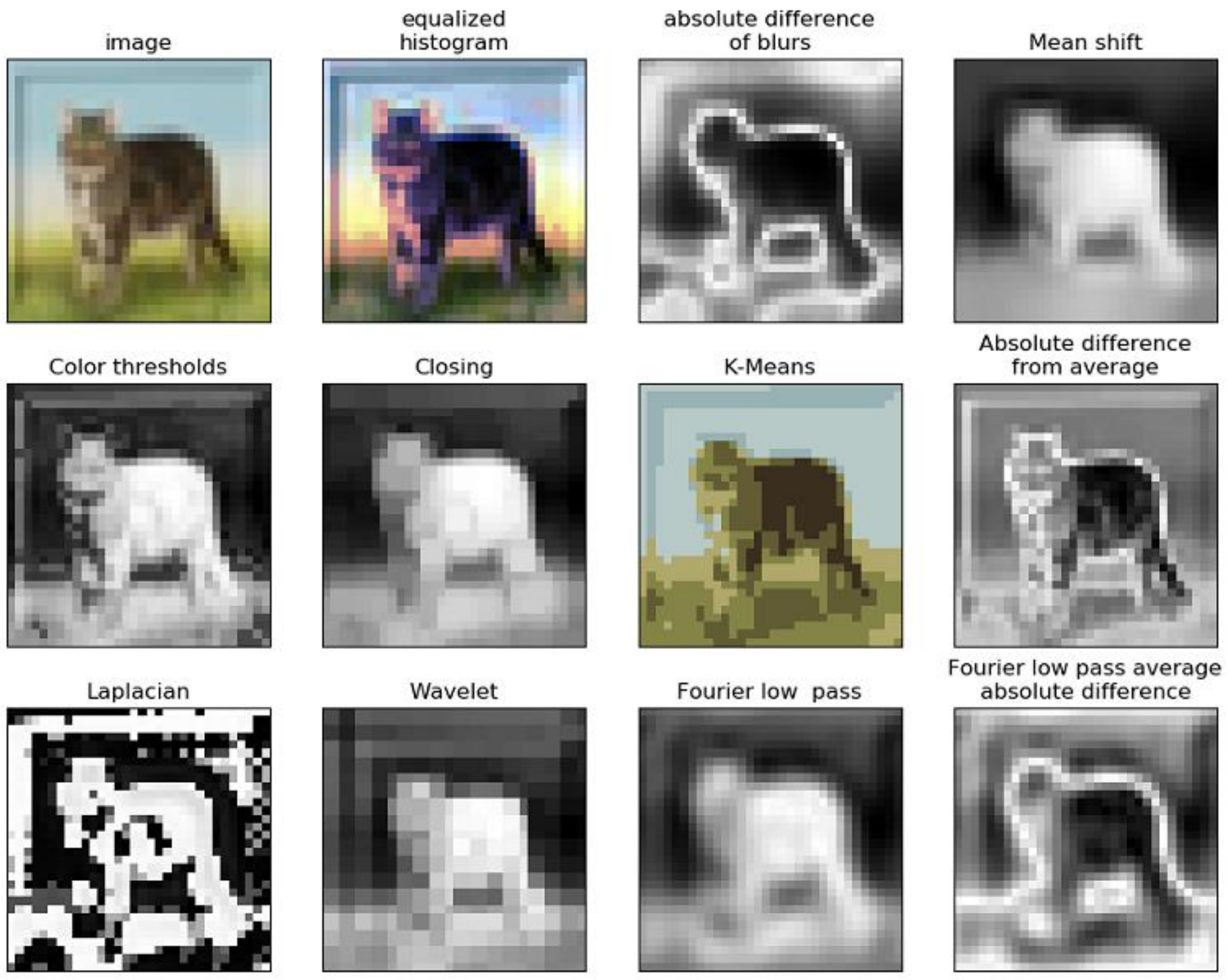
image
equalized histogram
absolute difference of blurs
Mean shift
Color thresholds
Closing
K-Means
Absolute difference from average
Laplacian
Wavelet
Fourier low pass
Fourier low pass average absolute difference

Figure 2. *Example filters applied to one of the Cifar10 images*

The accuracy result depends from the nature of the added filters, I tested up to ten filters but each time you add filters you increase the memory that is needed to perform the training. In all cases I got a better result from the initial network, and always got a faster training and validation increase rate even with one filter. I tested a combination that raised the initial network validation accuracy up to 93.6 % which is very good for a network less than 100 layer deep compared to the figures from Keras Resnet [KER 19]

Here is the compared learning on 200 Epochs from the base network and the network with four filters above with a training on a PowerAI machine using only one GPU [IBM 19]:

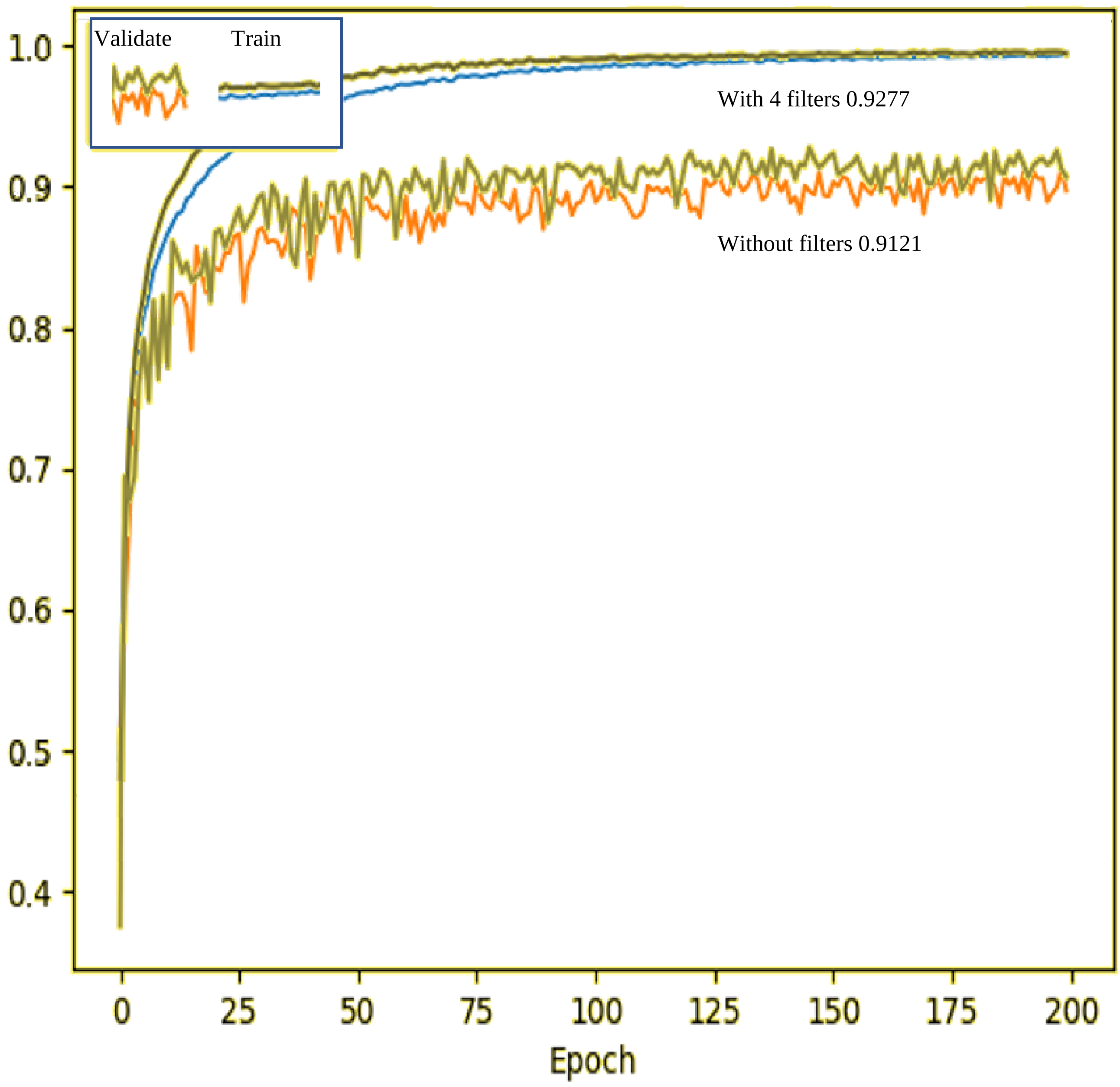

Figure 3. *Improvement from filter addition after 200 Epochs*

4. Conclusion

The above case with Cifar10 shows a 1.5% gain compared to the base network, however I got even much more significant gains with a project where the objects training images had more surrounding context and the resolution was bad. The selection of filters and the trial iterations require a fast machine so that in one day several filter options can be tested. I found that Wavelet Neural Networks WNNs had been tested a decade ago but my understanding is that an efficient backward propagation did not result from the trials. It would be good if the addition of non-differentiable filters, that I intuitively feel a cause of the improvement, could be mathematically proven.

Bibliography

[OLA 18] UNDERSTANDING RESIDUAL NETWORKS https://towardsdatascience.com/understanding-residual-networks-9add4b664b03

[KER 19] Keras Cifar Resnet https://keras.io/examples/cifar10_resnet/

[IBM 19] IBM PowerAI https://developer.ibm.com/linuxonpower/deep-learning-powerai/

[HON 08]Wavelet neural network for 2D object classification https://ieeexplore.ieee.org/abstract/document/4518022/