Preprocessing operators for image compression using cellular neural networks

Citation for published version (APA):

DOI:
10.1109/ICNN.1996.549122

Document status and date:
Published: 01/01/1996

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

Download date: 08. Nov. 2023
Preprocessing Operators for Image Compression Using Cellular Neural Networks

Oscar Moreira-Tamayo and José Pineda de Gyvez
Department of Electrical Engineering, Texas A&M University.
College Station, Texas 77843, USA. Tel.: (409) 845-7477, FAX: (409) 845-7161
Email: moreira@eesun.tamu.edu, gyvez@pineda.tamu.edu, URL: http://amesp02.tamu.edu

Abstract

Cellular Neural Networks (CNN) have traditionally been used to perform nonlinear operations on images such as edge detection, hole filling, etc. However, algorithms for image compression using CNN have scarcely been explored. This paper presents new templates and novel algorithms to perform basic operations used for image compression. They include wavelet subband decomposition, computation of parameters for bit allocation, quantization and bit extraction. These algorithms are hardware oriented and exploit the massive parallelism provided by the CNN. Compression is an important and widely used operation in image processing. Therefore, the algorithms presented here expand the realm of CNN applications. This feature is especially important for the widespread use of CNN as a multiple purpose image processor.

I. Introduction

Most image compression algorithms consist of finding a representation for the image that decorrelates its contents. This decorrelated representation can be coded with a lower number of bits since it contains no redundancy. High compression ratios can be obtained since most images in practice are highly correlated. For instance, most of the energy contained in typical pictures is concentrated in the low frequency bands. Reported algorithms for compression include cosine transform (used for JPEG standard [1]), subband decomposition [2], wavelet transform [3], among others. To reduce the error on lossy compression during quantization, it is often necessary to have statistical information such as the mean, and some measure of dispersion such as the variance. The process of compressing an image consists typically of the following steps [4]:

1. **Image transformation.** The purpose of this step is to represent the image in a form easier to decorrelate, e.g. wavelet transform, subband coding, cosine transform, lapped transform, etc.

2. **Quantization.** This step is usually performed for lossy compression. It consists of representing the data with a reduced number of bits for a given maximum error. The transformation in step 1 usually decomposes the image producing subimages with statistical distributions such as Laplacian, Gaussian, etc. This information can be used for reducing the error in scalar quantizers by setting the proper decision levels.

3. **Bit allocation.** Different sections of the decomposed image may require different number of bits to represent it. This step consist of finding which combination of quantizers will produce the minimum error while satisfying a budget constrain. Different quantizers can be implemented with CNN. The Lloyd-Max quantizer [5, 6] is addressed in this paper.

4. **Entropy coding.** Consist of coding the bit stream to increase the compression ratio, e.g. Huffman coding. This step is not performed through CNN.

II. Background on Cellular Neural Networks

For the following discussion consider a conventional [7] two-dimensional CNN array containing M×N locally connected cells. Any cell on the ith row and jth column, C(i,j) is connected only to its neighbor cells. The CNN dynamics are

\[
C \frac{dx_{ij}(t)}{dt} = - \frac{x_{ij}(t)}{\tau} + \sum_{C(k,l) \in N(i,j)} A(i,j;k,l)x_{kl}(t) + \sum_{C(k,l) \in N(i,j)} B(i,j;k,l)u_{kl} + I
\]  

(1)

---

1 Supported by CONACyT, the National Science and Technology Council of Mexico.
2 Research supported by the Office of Naval Research under contract grant No. N00014-91-1-0516

0-7803-3210-5/96 $4.00 ©1996 IEEE 1500
The neighborhood is denoted as \( N(i,j) \). Each cell has a state \( x \), a constant external input \( U \), and output \( y \). The first order nonlinear differential equation defining the dynamics of a cellular neural network cell can be written with general solution shown in eq.(3), where \( \Delta t \) is the time interval in which the CNN is operated, e.g., \( \Delta t=RC \).

\[
x_y(t) = \frac{1}{2} \left( |x_y(t) + 1| - |x_y(t) - 1| \right)
\]

The proposed strategy to implement linear operators is to zero the \( A \) template and set the current \( I=0 \). This will give the effect of using the CNN as a parallel convolutional operational array. Recall that filtering in the frequency domain is equivalent to performing a convolution in the time domain. Under these constraints, the steady state of eq. (3) reduces to eq. (4) which is a convolutional operation over the entire image space.

\[
x_{ij} = R \left( \sum_{C(k,l) \in N(i,j)} B(i,j;k,l)u_{kl} \frac{\Delta t}{RC} \right)
\]

Notice that \( x_y \) corresponds to the state of the cell and that its amplitude is not limited as \( y_y \) in eq. (2). In the remaining parts of the paper, it will be assumed that the input image has been normalized to a maximum value of 1 and a minimum of -1.

### III. Image Transformation

Common discrete wavelet transform basis for the one dimensional case are the Haar wavelet and the Daubechies wavelets [8]. We can obtain the wavelet decomposition shown in Fig. 1 by taking each row and then each column of the picture and performing a circular convolution with the scaling and wavelet functions. In Fig. 1 \( \textbf{G} \) and \( \textbf{H} \) represent a low pass and a high pass filter, respectively. They can be implemented through quadrature mirror filters or wavelet basis. In the last case \( \textbf{G} \) and \( \textbf{H} \) are the scaling function (\( \phi \)) and the wavelet function (\( \psi \)), respectively. The results are denoted as LL, LH, HL, and HH, where the first letter represents the operation performed in the horizontal direction and the second letter represents the operation performed in the vertical direction. This operation can be performed in one single step by obtaining the transformation matrices (2-dimensional transformation). The transformation matrices can be obtained by combining the scaling and wavelet function as shown in eq. (5).

\[
M_{LL} = \phi^T \phi , \quad M_{ LH} = \phi^T \psi , \quad M_{ HL} = \psi^T \phi , \quad M_{ HH} = \psi^T \psi
\]

For the case of a 4-point Daubechies wavelet, the scaling and wavelet functions are given by \( \phi = [c_0 \ c_1 \ c_2 \ c_3] \), and \( \psi = [-c_3 \ c_2 \ -c_1 \ c_0] \), where \( c_0 = [1+\sqrt{3}] / 4\sqrt{2} \), \( c_1 = [3+\sqrt{3}] / 4\sqrt{2} \), \( c_2 = [3-\sqrt{3}] / 4\sqrt{2} \), \( c_3 = [1-\sqrt{3}] / 4\sqrt{2} \).

The size of the transformation matrices is 4x4, e.g., the \( M_{LL} \) matrix is shown in (6)

\[
M_{LL} =
\begin{bmatrix}
c_0c_0 & c_0c_1 & c_0c_2 & c_0c_3 \\
c_1c_0 & c_1c_1 & c_1c_2 & c_1c_3 \\
c_2c_0 & c_2c_1 & c_2c_2 & c_2c_3 \\
c_3c_0 & c_3c_1 & c_3c_2 & c_3c_3
\end{bmatrix}
\]

In order to implement this transformation with CNN we need to take into account that the CNN templates are equal for all cells. Therefore, we will assume that both the image's horizontal and vertical sizes are a factor of four. The reason for this is that we have a 4 point transformation and if the sizes are not a factor of four there will be some rows and columns at the end of the array that will not match the corresponding ones in the transformation matrix. On the other hand, most CNN hardware implementations have a radius of 1, that is, each cell has connections only with its nearest neighbors. Therefore, in order to fit the transformation matrix into the 3x3, template the transformation process will be divided in four steps. That is, the transformation matrix is divided in four parts to yield the CNN templates necessary in each step. The proposed templates for the LL matrix are shown in (7).
The same process is used to obtain the templates $B$ for bands LH, HL and HH. In the first step, the image is provided as input without modification. In the second step we provide as input the image but shifting the position of all pixels by two places in the left direction and filling the right most two columns with the left most original ones (we are performing a circular convolution). In the third step we provide as input the original image but shifted up by two pixels as in the previous step and with the bottom most two rows filled with the first two original rows. In the fourth step we repeat the previous step but shifting in both vertical and horizontal directions. With these four steps every cell accumulates the necessary 4x4 input pixel array. Finally, the results are decimated and re-scaled by taking only the pixels with odd indexes, i.e., (1,1), (1,3), (3,1), etc. Fig. 1 shows simulation results for a Daubechies wavelet decomposition. The simulation results were obtained by building a CNN array through a Matlab [9] program.

![Image](image.png)

**Figure 1.** Wavelet decomposition scheme and simulation results. (a) Single Stage Image Decomposition, (b) results of decomposition using Daubechies wavelet. Clockwise: LL, LH, HL, and HH.

The reconstruction process is similar. The Daubechies wavelet is bi-orthogonal, therefore the reconstruction basis can be obtained by reversing the decomposition sequence and by alternating the signs of the coefficients. Then a new set of transformation matrices is obtained (in this paper denoted as $\tilde{M}$). In the reconstruction case equations (8) are implemented. More details and the step by step algorithms can be found in [10].

$$
\begin{align*}
&\tilde{p}_{i,j} = C_{LL,i,j} \cdot \tilde{M}_{LL} + C_{LH,i,j} \cdot \tilde{M}_{LH} + C_{HL,i,j} \cdot \tilde{M}_{HL} + C_{HH,i,j} \cdot \tilde{M}_{HH} \\
&\tilde{p}_{i+1,j} = C_{LL,i,j} \cdot \tilde{M}_{LH} + C_{LH,i,j} \cdot \tilde{M}_{HL} + C_{HL,i,j} \cdot \tilde{M}_{HH} + C_{HH,i,j} \cdot \tilde{M}_{LL} \\
&\tilde{p}_{i,j+1} = C_{LL,i,j} \cdot \tilde{M}_{HL} + C_{LH,i,j} \cdot \tilde{M}_{HH} + C_{HL,i,j} \cdot \tilde{M}_{LL} + C_{HH,i,j} \cdot \tilde{M}_{HH} \\
&\tilde{p}_{i+1,j+1} = C_{LL,i,j} \cdot \tilde{M}_{HH} + C_{LH,i,j} \cdot \tilde{M}_{HH} + C_{HL,i,j} \cdot \tilde{M}_{HH} + C_{HH,i,j} \cdot \tilde{M}_{HH}, \ \forall \quad i=1,3,...,M \quad \text{and} \quad j=1,3,...,N.
\end{align*}
$$

IV. **Computation of Statistical Parameters for Quantization**

The purpose of obtaining the mean and variance of a picture is to reduce the quantization error. A uniform quantizer has equally spaced decision levels. The Mean Squared Error (MSE) can be reduced by assigning non linearly spaced decision levels in such way that all levels have the same probability (this quantizer is known as Lloyd-Max quantizer). After decomposing an image, the elements LH, HL, HH (see Fig. 1) typically have a Laplacian distribution which can be described by its parameters: mean ($\mu$) and variance ($\sigma^2$). Therefore, it is important to calculate these parameters in order to have an effective quantizer. The proposed algorithm for finding the mean of an image consists of several steps depending on the size of the picture. Since each cell has connections only with its
nearest neighbors, the average of at most nine pixels can be taken at a time. The results of the means of each nine pixel block can be reprocessed until one cell accumulates the total result. Computing the average of all nine cells has a similar problem as with the decomposition. If the pixels of the input picture are not arranged in such a way that the number of pixels on the sides are a multiple of three, there will be unmatched rows and columns at the ends. That is, cells at the end of the array will have less than nine connected input pixels. Yet, templates must remain the same for all pixels. Since we are dealing with wavelet transform decompositions in which the basis is dyadic, it is therefore convenient to assume that both the horizontal and vertical sizes of the input image are a power of two. For this case the templates to compute the average of four adjacent pixels (called a block) are the following:

\[
A = [0], \quad I = 0, \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1/4 & 1/4 \\ 0 & 1/4 & 1/4 \end{bmatrix}
\]

(9)

It can be seen that if the CNN array runs for a time \( \tau = RC \), a cell \( C_0 \) will accumulate the average of four neighboring input pixels (\( u_i, u_{i+1}, u_{i+j}, u_{i+j+1} \)). Only one output pixel value is needed for each block, e.g., the output of the upper left cell in each 4 pixel block. These local averages are then fed as inputs and the process is repeated until one cell accumulates the final mean value. The size of the input array is one fourth with respect to the previous iteration since we are taking only one output for each block. If the size of the image is \( 2^M \) by \( 2^M \) then the number of iterations is \( M \). It can be seen here that an image of size in powers of two is convenient. In each iteration both the horizontal and vertical sizes are divided by two, but the size of the output array in each iteration will remain as a power of two.

The process of calculating the variance involves obtaining the expected value of the squared value of the pixel minus the mean \( \sigma^2 = E[(u - \mu)^2] = E[u^2] - \mu^2 \). Implementing the squared value is not simple. However, the non-linearity provided at the output of the CNN array can be exploited to compute the norm 1 of the image \( \| u - \mu \|_1 = E[|u - \mu|] \).

This is done by calculating the mean of the absolute value of the image. For the Laplacian distribution, \( \sigma^2 \) can be estimated from this norm as

\[
E[|u(x) - \mu|] = \int_{-\infty}^{\infty} |u(x) - \mu| f_u(x) dx = \int_{-\infty}^{\infty} |u(x) - \mu| \sqrt{2 \pi} e^{-\frac{(|u(x) - \mu|)^2}{2}} dx = \frac{\sqrt{2}}{2} \sigma
\]

therefore,

\[
\sigma^2 = 2E^2[|u(x) - \mu|]
\]

(10)

(11)

Using this equation the quantization tables for the Lloyd-Max quantizer can easily be updated. Table 1 shows the decision and reconstruction levels for a 2 bit quantizer based on the Laplacian distribution [11]. The reconstruction level is the value given to the elements that belong to a quantization level during reconstruction. These reconstruction levels minimize the mean square error.

<table>
<thead>
<tr>
<th>Bits</th>
<th>decision levels ( d_i )</th>
<th>reconstruction levels ( r_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(-\infty, -1.1269, 0.0000, 1.1269)</td>
<td>(-1.8340, -0.4198, 0.4198, 1.8340)</td>
</tr>
</tbody>
</table>

The norm 1 can be computed using CNN by separating the positive and negative parts first. The positive part of the absolute function can be obtained by performing the following steps. First initialize the CNN array in the saturation limit, i.e., \( x_\phi(0) = -1 \). Second, set the CNN parameters as

\[
I = -\mu, \quad A = [0], \quad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}
\]

(12)

then run the network for a time \( \Delta t = RC \). Observe that the pixel values will be displaced from their [-1,1] normalized range to \([-1-\mu, 1-\mu]\). The CNN array will rectify the input image, since negative terms in \( u_i - \mu \) are truncated as illustrated in Fig. 2. Here, the range of the pixel values is represented by the rectangle. Points \( E \) and \( F \) are the limits of the linear output range for this particular example. Values below the mean will become -1 because of the limiter function (eq. (2)). To eliminate the biasing introduced by the initialization of the CNN \( x_\phi(0) = -1 \), the templates
$l=1, A=[0], B=[0]$ are used for a time $\Delta t=RC$. This will effectively add 1 to the image values. The range of the image now goes from 0 to its maximum value. To separate the negative terms in $u_{ij} - \mu$ a similar operation is performed. The network is now initialized in the positive saturation limit, i.e., $x_i(0) = 1$, and the same templates (12) are used. This will leave in the network only the elements below the mean since those above the mean are clipped by (2). The bias is compensated again by running the array with $I=1$. Finally the first image (positive values) need to be subtracted from the second image (negative values) to produce the absolute value of the image. This operation can be performed by initializing the array with the first image, providing as input the second image, and using the templates (12) with the center value of $B$ equal to -1. The norm $1 (E|lu_{ij} - \mu|)$ is just the mean of this absolute value image and can be obtained with the previously explained procedure. Fig. 3 shows a rectified image processed by using the algorithm just described. Its denormalized parameters are the following, the mean is $\mu = 99.1179$ and the variance is $\sigma^2 = 2 (43.8136)^2$.

![Diagram](a) Fig. 2 Results for statistical computations. (a) Computation of the absolute value using the non-linearity of the CNN output (see eq. (2)). (b) Absolute value of input image.

V. Quantization

The purpose of quantization is to represent the image with a lower number of bits. The objective of a Lloyd-Max quantizer is to divide the signal range in a discrete number of segments in such way that each segment has equal probability (this will minimize the MSE). This section will address the problem of assigning the pixels to the segment they belong. If one were to represent an image with one bit per pixel we have two segments. Positive values are represented with logic 1 and negative values with logic 0. The image can be just thresholded with respect to its mean to obtain two output levels, 1 and -1 for the CNN array. This can be done by making $I=\mu$ and $A=B$ with $B$ as in (12) and taking the output from $y_{ij}$. The network will be allowed to run until the output values reach their limits.

For a quantizer consisting of $n$ decision levels $d_n$, the level to which any particular pixel belongs can be extracted using the following recursive process. Start by the highest decision level $d_1$ (for 2 bits $d_2 = 1.1268$ from Table 1, see Fig. 3). Set $I=d_n$ , and again $A=B$ with $B$ as in (12). The bit extraction process consists in keeping those cells whose $y_{ij}$ outputs are 1. These pixels belong to the first segment. Consider now the next decision level $d_{n-1}$ (for 2 bits $d_2 = 0.0$). If we use the same input picture the pixel levels belonging to this segment will be confused with those of the previous iteration (both segments have values above the threshold level). Therefore, it is needed to eliminate the pixels that presented output 1 in the previous iteration. To do this we subtract the output of the previous iteration from the input picture. This will set the pixels higher than $d_n$ to a value $d_n - 1$, while the pixels lower than $d_n$ will become $d_n + 1$. Last, return pixels to its original biasing by running the CNN with templates $I=1, A=B=[0]$. Repeat this process for $d_{n-1}, d_{n-2},$ etc.

Once the bits are extracted we can reconstruct the image from the binary data. This is an interesting feature since it implies that input data can be provided in a digital form as well as in the existing analog form. Suppose that we have the reconstruction levels associated to the Lloyd-Max quantizer $r_i$ and that we have $n$ bits. To reconstruct the image we will use the following process. Feed the first bit of each pixel of the whole image as the input array, then set the template as in (13). By running the CNN for $\Delta t = RC$ the value of each weighted bit will be loaded in the CNN. This process is repeated for all remaining bits and the partial results are accumulated having at the end the image loaded in the CNN array. Notice that the D/A conversion is actually performed by multiplying the digital bit by its analog value $r_i$.
$l=0$, $A=0$, and $B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & r_l & 0 \\ 0 & 0 & 0 \end{bmatrix}$ (13)

Fig. 3b shows the result of quantizing the original image in the following way: The original image was decomposed using Daubechies wavelet in two levels (the first LL result was also further decomposed). 2 bits were allocated for the decomposition elements LH, HL, and HH on each level, and 8 bits were allocated for the second LL element.

Figure 3. Quantization. (a) Illustration of a 2 bit quantization levels. (b) Quantized and reconstructed image.

VI. Conclusion

Preprocessing algorithms for image compression have been developed. They exploit the massive parallelism that CNN provides. By adding an entropy coder to the output provided by this CNN algorithms fully compressed images can be obtained. Entropy coding is not implemented thru CNN because it is not an image transformation but a digital coding that involves the manipulation of individual bits on the bit stream. The compression algorithms given in this paper are also suitable for multiplexing CNN arrays [12].

VII. References


1505