THE ONE-TIME LEARNING HIERARCHICAL CMAC AND THE MEMORY LIMITED CA-CMAC FOR IMAGE DATA COMPRESSION

Ted Tao, Hung-Ching Lu*, Chau-Yun Hsu, and Ta-Hsiung Hung

ABSTRACT

Two methods to compress transmitted image data are proposed in this paper. The first method is the one-time learning hierarchical CMAC method and the second is the memory limited CA-CMAC method for image data compression and reconstruction. The one-time learning hierarchical CMAC method is used when a coarse image needs to be sent to the receiver initially and then the image quality is gradually improved at the request of the receiver. But, when the transmitting channel data is limited, the memory limited CA-CMAC method can be used to decrease the bit rate per pixel. Both proposed methods, unlike conventional compression methods, use no filtering technique in either compression or reconstruction. CMAC networks use a few hypercubes to learn the characteristics of the original image, so image data can be compressed without suffering from blocking effect or boundary effect. One-time learning is good enough for compressing image data, and it has a high SNR after reconstruction.

Key Words: CMAC, CA-CMAC, image, compression and reconstruction.

I. INTRODUCTION

Network transmission has become increasingly popular, and image data transmission is used very frequently in network transmission. The volum of images transmitted data and reconstruction time are the most important things that we should consider. We propose two methods to compress transmitted data, which can save a lot of memory and transmitting time. The first method is the one-time learning hierarchical CMAC (cerebellar model arithmetic computer) method (Albus, 1975a; Albus, 1975b; Iiguni, 1996) and the second is the memory limited CA-CMAC (credit assigned cerebellar model arithmetic computer) method (Smalz and Conrod, 1994; Su et al., 2002) for image data compression and reconstruction.

The one-time learning hierarchical CMAC method is used when a coarse image needs to be sent to the receiver initially and then the image quality is gradually improved at the request of the receiver. In other words, the receiver recognizes the image content at an early stage, and then decides whether further transmission is necessary or not. Consequently, transmission time can be saved. When the transmitting channel data is limited, the memory limited CA-CMAC method can be used. It can reduce the bit rate per pixel and the number of additions per pixel during the reconstruction process.

A number of transforms such as DCT (Mukherjee and Mitra, 2002; Rao and Yip, 1990), DST (Yip and Rao, 1980), Fast Karhunen Loeve transform (KL) (Jain, 1976), and wavelet transform (WT) (Arerbuch et al., 1996) have been used to compress image data. They all have blocking effect in low-resolution images and need filters that require lots of computations. To overcome the above problems, we propose two methods to compress transmitted data. CMAC is an associative memory neural network based on a table look-up method. There are several advantages including local generalization (Gonzalez-Serrano et al., 1998), good learning capacity, and rapid learning convergence, all of which
have been demonstrated in reference (Wong and Sideris, 1992; Lin and Chiang, 1997), and are easy to implement in automatic control (Miller et al., 1990; Chen and Chang, 1996) and signal processing (Patra et al., 1999; Hong et al., 1997), etc.

However, when the conventional CMAC approach, as used in the literature, is employed, the learning still needs several cycles, or epochs, to converge. It is not good enough for one-time learning. Recently, several approaches have been proposed to improve the learning performance of CMAC, such as reference (Chiang and Lin, 1996; Lin and Chiang, 1998). Most of them introduced the concept of various weights into the cell structure of CMAC. As expected, the kind of interpretation capability indeed can increase the accuracy of the representation of the stored knowledge. However, the speed of convergence still cannot satisfy the requirement for one-time learning applications.

In this paper a novel learning scheme is proposed to speed up the learning process so that one-time learning can be possible. In the conventional CMAC learning schemes, the correct amounts of errors are equally distributed into all addressed hypercubes regardless of the credibility of those hypercubes. Such an updating algorithm violates the concept of credit assignment (Smalz and Conrad, 1994), which requires that the updating effects be proportional to the responsibilities of hypercubes. The proposed learning approach is to use the inverse of learned times of the addressed hypercubes as the credibility (confidence) of the learned values; we call this the credit assigned CMAC (CA-CMAC) method. With this modification, the learning speed can indeed become very fast. In our implementation, this approach can save a lot of time and memory. With our proposed credit assigned CMAC, the memory limited CA-CMAC can actually learn original image data with very low root mean square error (RMSE) and high signal to noise ratio (SNR).

The other sections of this paper are organized as follows. The CMAC structure is introduced in Section II. One-time learning hierarchical CMAC for image data compression and reconstruction is presented in Section III. The memory limited CA-CMAC is deduced in Section IV. Treatment of boundary effects and the convergent condition of CA-CMAC are discussed in Section V. The simulation results in Section VI and comparisons in Section VII show the excellent one-time learning capability of the proposed CA-CMAC for image data compression and reconstruction. Finally, Section VIII concludes the paper.

II. CMAC STRUCTURE

We will introduce the structure and learning algorithm of a two-dimensional CMAC in order to show image data learning in this section. The structure of the two-dimensional CMAC is shown in Fig. 1. The input vector is defined by two input variables $x$ and $y$. In this example, 17 states are to be distinguished for each variable and four floors are used for each variable. In the first floor, $x$ is divided into 5 blocks from $A$ to $E$, and $y$ is also divided into 5 blocks from $a$ to $e$. There are 25 locations from $Aa$ to $Ee$, which can store data on the first floor. Such locations are often called the hypercubes. We shift one state on the second floor, shift two states on the third floor, and shift three states on the fourth floor. Then, there are 25 different hypercubes on each floor. Be aware that only the blocks on the same floor can be combined to form a hypercube. In other words, the hypercubes such as $Af$ and $Fk$ do not exist. Therefore, there are 100 hypercubes in this two-dimensional CMAC structure. When each input vector is specified, four hypercubes from the four floors are addressed. For example input $(9,9)$ maps to $Cc$, $Hh$, $Mm$ and $Rr$ four hypercubes. Note that these 100 hypercubes can be used to distinguish 289 different pixels. We also find that each hypercube shares data with neighboring pixels, so there will not be any blocking effect.

In general, let $(x, y)$ be the considered pixel and $w_{(x,y)}(j)$ be the stored data in the $j$th floor hypercube which is mapped by pixel $(x, y)$, and the number of the floors of CMAC be $K$. Then the output value $g(x, y)$ can be computed as indicated in the following equation:
\[ g(x, y) = \sum_{j=1}^{K} w_{(x,y)}(j) \]  

Since each pixel addresses exactly \( K \) hypercubes, only those data are used in the Eq. (1) for each pixel in the CMAC.

Because the output value \( g(x, y) \) is generated from the CMAC hypercubes, the data of hypercubes need to be modified by the error \( e(x, y) \) that is the difference between the destined value \( d(x, y) \) and output value \( g(x, y) \) in the learning process. The updating algorithm for \( w_{(x,y)}(j) \) is defined by the following equations:

\[ w_{(x,y)}^{\text{new}}(j) = w_{(x,y)}^{\text{old}}(j) + \frac{u}{K} e(x, y); \quad j = 1, 2, \ldots, K \]  

\[ e(x, y) = d(x, y) - g(x, y) \]  

Where \( u \) is a learning rate in the learning process, and according to reference (Lin and Chiang, 1997) if \( 0 < u < 2 \) is true, then the above learning algorithm will converge.

Suppose that there are \( N \) states to be distinguished for each dimension, and \( K \) floors in CMAC. Then there are \([\text{ceil}((N+j-1)/K)]^2\) hypercubes in the \( j \)-th floor, where the function \( \text{ceil}(x) \) rounds the elements of \( x \) to the nearest integer towards infinity. Thus the total number of hypercubes in a CMAC is described in the following equation:

\[ N_m = \sum_{j=1}^{K} \{\text{ceil}((N + j - 1)/K)\}^2 \]  

It is seen that there are only \( N_m \) hypercubes needed to distinguish \( N^2 \) pixels. The significant property of CMAC is that the learning algorithm changes the output values for the nearby inputs. Therefore similar inputs lead to similar output even for untrained inputs. This property is called generalization, which is of great use in the CMAC based coding. Moreover, we can control the degree of generalization by changing the size of \( K \). The larger \( K \) is, the wider the generalization region is. The generalization region of a CMAC with \( K=4 \) is shown in Fig. 1, and a training pixel \( d(9,9)=4 \) is given as the input. The input data maps to four hypercubes denoted by Cc, Hh, Mm and Rr, and four weights stored in the hypercubes are updated by Eqs. (1) and (2). Let the initial values of all weights be zero, then the error \( e(9,9) \) equals 4 before learning. The neighborhood of the pixel \((9,9)\) including itself will be all updated, and their values are “1”, “2”, “3”, and “4”. More precisely, the outputs indicated by “4”, “3”, “2”, and “1” are updated by \((ue), (3ue/4), (ue/2), \) and \((ue/4)\), respectively, and \( e \) is the error \( e(9,9) \) which is the difference between the destined value \( d(9,9) \) and output value \( g(9,9) \) before the updating. We find that the neighborhood outputs are changed even for untrained input after updating by the CMAC learning algorithm. The generalization property of CMAC will be used to compress image data in later sections.

### III. ONE-TIME LEARNING HIERARCHICAL CMAC METHOD FOR IMAGE DATA COMPRESSION AND RECONSTRUCTION

If we only use one CMAC structure for one-time learning, the data in the neighboring pixels will interfere with each other. The reason is that the error equally distributes to each hypercube by Eq. (2), so the latter training pixel may corrupt the former data in selected hypercubes, which already contain some knowledge of previous training. Thus, the previously learned information may be corrupted due to errors caused by neighboring pixels. In order to prevent the interference phenomenon, we have to divide the CMAC into \( m \) independent segments, whose data will not interfere with each other. So we propose a process of only one-time learning hierarchical CMAC for image data compression, whose block diagram is shown in Fig. 2. It is seen that the memories are divided into \( m \) segments, and each segment uses an independent CMAC(i) structure, whose weights of hypercubes in CMAC(i) shall not interfere between
different segments. Let the $i$-th segment of CMAC (i) include $K_i$ floors, and if every pixel uses one byte data then the bytes of memories used in $i$-th segment $N_i$ can be computed by the following equation:

$$N_i = \frac{\text{ceil}(N + j - 1/K_i)^2}{m};$$

$$K_i = 2^{m-i}; i=1, 2, ..., m-1$$ (5)

where $N$ is the number of pixels in each dimension.

1. The Compression Process

After proposing the diagram of only one-time learning hierarchical CMAC, we need to encode the input data in the compression process. In view of the diagram, the $Q$ block is considered as the selecting and quantizing process. Through this process the symbol $e_i$ becomes $\tilde{e}_i$. When we select the training pixels for $i$-th segment ($i=1$ to $m-1$) as training data, the distance between every selected point is $2^{m-i}$ points along each dimension. If there are $N x N=2^n x 2^n$ pixels in the original image, there will be $2^{2(n-m-i)}$ data trained in the $i$-th segment. To prevent the data overlapping each other in the compression process, we add a shift vector to the training pixel. Shift vector equals (1,1) in the first segment, and the shift vector equals ($2^{m-i}+1, 2^{m-i}+1$) in the $i$-th segment ($K \times i = 2$) segment. Not all the ordered pixels should be trained according to sample density. The other no-order pixels, which have not been denoted, can be trained in the last order to increase SNR or need not be trained to save computation. Note that we may directly select the errors of every $2^{m-i}$ pixel along each dimension and their shift vectors are (1,1) added to the last segment outputs. We take the 17x17 size image as an example, and let the number of segments be $m=5$. Then, the training order of the image is shown in Fig. 3, where the selected pixels are denoted by “*” in the $i$-th segment, because the selected pixels are every $2^{m-i}$ points along each dimension in the $i$-th segment, and the $i$-th CMAC(i) has $2^{m-i}$ floors. It is impossible for them to interfere with each other during the compression process, because every hypercube is selected just one-time. Because each pixel maps to $K_i$ hypercubes, so the nearby pixel can be generated even it hasn’t been trained. This is the reason why the learning algorithm needs only one-time learning. It is described in the following equations:

$$e_i(x, y)=d(x, y)$$ (6)

$$e_i(x, y)=e_{i-1}(x, y)−g_i(x, y)$$ (7)

$$g_i(x, y) = \sum_{j=1}^{K_i} w(x, y, j)$$ (8)

$$w^{new}_{(x, y, j)}(i) = w^{old}_{(x, y, j)}(i) + \frac{N}{K_i} \tilde{e}(x, y); K_i = 2^{m-i}$$ (9)

There exists one division in Eq. (9). But the computation time can be ignored, when the number $K_i$ is a power of 2 and it is performed with a shift operation. From Eqs. (7) to (9), we notice that the number of additions per pixel in the $i$-th segment equals $2/K_i$. Then, the number of total additions per pixel $N_{add}$ in all segments is described as $N_{add}=2^{m-i} \sum_{i=1}^{m-1} \frac{1}{K_i}$. If the size of the image is selected as $256 \times 256$ and $m$ is chosen as 5, it needs only 1.875 additions per pixel in the fourth segment. With so few computations, it takes very little time per pixel during the compression process.

2. The Reconstruction Process

The reconstruction process may be considered as a decoding process, and it is described in the following equations:

$$h_i(x, y)=g_i(x, y)$$ (10)

$$h_i(x, y)=\tilde{e}_{i-1}(x, y)+h_{i-1}(x, y); i=2, 3, ..., m$$ (11)

From Eqs. (10) and (11), we can compute that there are $2^{m-i} x 2^{2n}$ additions in the first segment, $2^{m-i} x 2^{2(n-m-i)}$ additions in the $i$th ($i=2, 3, ..., m-1$) segment, and $2^{2(n-m+2)} x 2^{2(n-m+1)}$ additions in the last segment where $2^{2(n-m+1)}$ is the number of selected pixels both
in the first and last segment. If the size of the original image is 256x256, the number of required additions will be 15.8867 per pixel in the last segment reconstruction. It is obvious that the number of additions per pixel in the last segment reconstruction is more than that in the compression process. Now computing the bit rate per pixel \( H(b/\text{pixel}) = \frac{1}{N^2} \sum_{i=1}^{N^2} (2^{n(2m+2)}-2^{n(m+1)}+\sum_{i=1}^{N_i}) \) in the last segment, it equals 0.9735 (b/pixel) if \( N^2=256^2 \) and \( m=5 \).

Now the error after reconstruction is discussed and it can be computed by using the following equation:

\[
e_0(x, y) - h_m(x, y) = e_{m-1}(x, y) - \tilde{e}_{m-1}(x_m, y_m) \quad (12)
\]

The more untrained pixels errors \( e_{m-1}(x, y) \) selected as \( \tilde{e}_{m-1}(x, y) \) added to the \( m \)-th segment outputs, the less errors are in the last segment. If all errors of untrained pixels are selected, it means that the pixels of \( \tilde{e}_{m-1}(x, y) \) equal those of \( e_{m-1}(x, y) \), and the error of output in the last segment nears zero. The cost of lower errors is that the bit rate per pixel becomes higher. For example, if all errors of untrained pixels are transferred, the bit rate per pixel \( H \) (b/pixel) becomes 1.6298 for \( N^2=256^2 \) and \( m=5 \).

From the analysis of the bit rate and computations of \( N^2 \) pixels in \( m \) segments, there are two advantages to one-time learning hierarchical CMAC for image data compression. The first advantage is that it takes very few computations. For instance, if the size of the original image is 256x256 and \( m=5 \), it needs only 1.9236 additions per pixel during the compression process. The second advantage is that it takes only one-time learning in the compression process. However the bit rate per pixel is still high and it needs more additions (e.g. it needs 16 additions in the first segment of reconstruction for \( m=5 \)). And these are the main drawbacks during the reconstruction process. Thus, we will propose the memory limited CA-CMAC method to reduce the bit rate per pixel and the number of additions per pixel during the reconstruction process in the next section.

IV. THE MEMORY LIMITED CA-CMAC METHOD

If we want to increase SNR, we have to increase bit rate and use more computations in the one-time learning hierarchical CMAC method for image data compression and reconstruction. So we propose the memory limited CA-CMAC method to decrease the number of additions and the bit rate per pixel during the reconstruction process, while SNR is still high.

In order to decrease the bit rate per pixel, we use only one CMAC structure that contains \( K \) floors. The training order of selected pixels is the same as the one-time learning hierarchical CMAC method, so the bit rate per pixel can be reduced to \( N_e/N^2 \). And if \( N=256 \) and \( K=4 \) then \( N_e=16771 \), and the bit rate per pixel becomes 0.2559 (b/pixel) which is near 1/4. Obviously, it is lower than 0.9735 (b/pixel) in the one-time learning hierarchical CMAC method.

Although its bit rate per pixel is low, the error will be high if it is still one-time learning. The reason is that the error distributes to each hypercube, which is selected in the above learning process. The latter training pixel may corrupt the former data in selected hypercubes, which already contain some knowledge of previous training. Thus, the previous learned information may be corrupted due to errors caused by neighboring pixels. Therefore, it needs more iteration to smooth the corruption when using one CMAC to compress image data. This is evident from successful learning in various CMAC applications (Thompson and Kwon, 1995). However, there may not be enough time for smoothing out the corrupted data. Thus, the learned results of the updating algorithm may not be acceptable. Hence, a novel learning approach embedding the concept of credit assignment is proposed to account for the previous learning behavior, and the results have indeed justified our claim and demonstrated good learning properties. From our simulation, it can be seen that the learned results will converge very quickly even with only one-time learning. In our research, such a learning scheme is called the credit assigned CMAC or CA-CMAC.

In order to avoid such corruption effects, the error correction must be performed according to the creditability of the hypercubes. Such a concept is often referred to as the credit assignment for learning. As we have stated, the distribution of errors into the addressed hypercubes must be proportional to the creditability of those hypercubes. However, in the CMAC learning process, there is no way of determining which hypercube is more responsible for the current error or is more accurate than the others. The only information that can be used is how many times the hypercubes have been updated. The assumption used in our approach is that the more times the hypercube has been trained, and the more accurate the stored value is, so it should get less updating. Hence, the times of updating for hypercubes can be viewed as the creditability of those hypercubes. With such an assumption, the term \( w_i(x, y) \) in Eq. (1) is replaced as follows:

\[
w_i(x, y)(j) = Kh_i(x, y)(j) V_i(x, y)(j)
\]

In which
\[ b_{i(x,y)}^{(j)}(y) = \frac{1}{f_{i(x,y)}(j) + 1} \times \left( \sum_{j=1}^{K} \frac{1}{1 + f_{i(x,y)}(j)} \right)^{-1} \]  

where \( f_{i(x,y)}(j) \) is the learned times of the \( j \)-th hypercube in \( i \)-th iteration mapped by the pixel \((x, y)\). The idea of the updating algorithm is that the effects of error correcting must be proportional to the inverse of the times of learning of those selected hypercubes. Since the times of learning must include the present time to prevent dividing by zero, the learning coefficient in Eq. (14) becomes \( \frac{1}{f_{i(x,y)}(j) + 1} \). If the error \( E \) is defined in Eq. (16) and the steepest rule is selected as \( \Delta V_{i(x,y)}^{(j)} = -\eta \frac{\partial E}{\partial V_{i(x,y)}^{(j)}} \) (\( \eta \) is learning constant), the learning algorithm will be deduced as follows:

\[ g_{i(x,y)} = \sum_{j=1}^{K} Kb_{i(x,y)}^{(j)} V_{i(x,y)}^{(j)} \]  

\[ E = \frac{1}{2} (d_{i(x,y)} - g_{i(x,y)}^{(j)})^2 \]  

\[ \Delta V_{i(x,y)}^{(j)} = -\eta \frac{\partial E}{\partial V_{i(x,y)}^{(j)}} = \eta Kb_{i(x,y)}^{(j)} (d_{i(x,y)} - g_{i(x,y)}^{(j)}) \]  

\[ V_{i(x,y)}^{(j)} = V_{i(x,y)}^{(j-1)} + \eta Kb_{i(x,y)}^{(j)} (d_{i(x,y)} - g_{i(x,y)}^{(j)}) \]  

From the above algorithm, the bit rate per pixel \( H = \frac{N_o N^2}{b} \) (b/pixel) is very low; for example, if \( K=4 \) then \( H=0.2559 \) (b/pixel). The cost is that we need to count the times trained for each hypercube, which takes very little time during the compression process. However, it only uses \( K \) additions per pixel, which is lower than the one-time learning hierarchical CMAC method during the reconstruction process, because the reconstructive equation is defined as the Eq. (19).

\[ g_{i(x,y)} = \sum_{j=1}^{K} V_{i(x,y)}^{(j)} \]

V. TREATMENT OF BOUNDARY EFFECTS

AND THE CONVERGENT CONDITION

OF LIMITED MEMORY CA-CMAC

1. Treatment of Boundary Effects

It is known that boundary errors will exist after reconstruction, if the boundary pixels are not trained. This is called the boundary effect (Iiguni, 1996) in CMAC learning. To explain this phenomenon, we take a one dimension CMAC as an example. There are 16 points along the \( x \)-axis, and the CMAC is divided into four floors in Fig. 4(a). If we select every four points along the \( x \)-axis as training data, that is 1, 5, 9, and 13, we find that the hypercube \( J, O, \) and \( T \) will not have any chance to be updated. Therefore, these hypercubes will cause errors when the boundary points 14, 15, and 16 are reconstructed.

To prevent boundary effects, we must select boundary points as training data. If the selected training data include boundary points, then boundary effects will disappear. In Fig. 4(b), the boundary point 17 is selected as a training datum, so every hypercube gets equal updating information. If there is a 256x256 size original image when the number of CMAC floors is selected as 4, we choose left boundary pixels because the shift vector is (1,1) in the first training order. Right boundary pixels are also selected as training data, since the shift vector is (2,2) in the third training order. This is the way to prevent boundary effects in the compression process.

2. The Convergent Condition of Limited Memory CA-CMAC

The convergent condition of conventional CMAC learning algorithm as described in Eq. (2) and (9) is the learning rate \( u \) needs to be in the range of \( 0 < u < 2 \). This has been proven in reference (Lin and Chiang, 1998), and it will not be discussed in this paper. However the convergent condition of limited memory CA-CMAC learning algorithm is \( 0 < \eta < \frac{2}{K^2 b_{i(x,y)}^{(j)}} \), which will be proven in this section.

Lemma 1:
If the sequence \{ \( x(i) \) \} has the property
\[
\frac{x(i) - x_d}{x(i) - x_d} = r_i \text{ for all } i \geq 1, \ |r_i| < 1 \text{ and } |x(0) - x_d| \leq M. \ M \text{ is a positive bounded real number, then the sequence will converge to } x_d. 
\]

**Proof:**

Define \( \Delta x_i = x(i) - x_d \), then \( r_i = \frac{\Delta x_i}{\Delta x_{i-1}} \) and it yields \( \Delta x_i = r_i \Delta x_{i-1} \). Next, let \( r = \max \{ |r_1|, |r_2|, \ldots, |r_i| \} \), then \( r \leq 1 \) is still true because every \( |r| < 1 \). Therefore, \( |\Delta x_i| \leq r^i |\Delta x_0| \) is true when \( i \) approaches infinity; then \( \Delta x_i \) approaches zero because \( |\Delta x_0| = |x(0) - x_d| \) is a bounded value. In other words, \( x(i) \) will converge to \( x_d \) as \( i \) approaches infinity.

**Theorem 1:**

If \( 0 < \eta < \frac{2}{K^2 b^{i-1} \left( x_{i-1} \right) (j)} \) is true, then the updating algorithm (18) will converge.

**Proof:**

The Eq. (18) can be rewritten as

\[
V_{(x, y)}^i(j) - V_d = V_{(x, y)}^{i-1}(j) - V_d + \eta K b^{i-1} \left( x_{i-1} \right) (j) \left( d_{(x, y)} - V_{(x, y)}^{i-1}(j) \right) - \frac{K}{b-1} K b^{i-1} \left( x_{i-1} \right) (j) V_{(x, y)}^{i-1}(j)
\]

and

\[
\text{For } i = 1, \text{ we can get } \Delta x_0 = x(1) - x_d = V_{(x, y)}^0(j) - V_d = \eta d - V_d \text{ which is a bounded value with initial value } V_{(x, y)}^0(j) = 0 \text{ and } b^{i-1} \left( x_{i-1} \right) (j) = 1/K. \text{ Now, we use } r_i = \frac{\Delta x_i}{\Delta x_{i-1}}, \text{ according to Lemma 1, the convergent condition is}
\]

\[
1 + \frac{\eta K b^{i-1} \left( x_{i-1} \right) (j) \left( d_{(x, y)} - \sum_{j=1}^{K} K b^{i-1} \left( x_{i-1} \right) (j) V_{(x, y)}^{i-1}(j) \right)}{V_{(x, y)}^{i-1}(j) - V_d} < 1
\]

for all \( i \geq 1 \); that is,

\[
0 < \eta K b^{i-1} \left( x_{i-1} \right) (j) \frac{K}{b-1} K b^{i-1} \left( x_{i-1} \right) (j) V_{(x, y)}^{i-1}(j) \frac{V_{(x, y)}^{i-1}(j)}{V_{(x, y)}^{i-1}(j)} < 2.
\]

Next, it will be discussed as the following three cases:

**Case (1):** \( i = 1 \)

For initial value \( V_{(x, y)}^0(j) = 0, b^{0} \left( x_{i-1} \right) (j) = \frac{1}{K} \) and \( V_d = \frac{d}{K} \), we can get

\[
\frac{1}{K} \left( d_{(x, y)} - \sum_{j=1}^{K} K b^{0} \left( x_{i-1} \right) (j) V_{(x, y)}^{0}(j) \right) = 1.
\]

Thus, the convergent condition \( 0 < \eta < \frac{2}{K^2 b^{i-1} \left( x_{i-1} \right) (j)} \) is gotten for \( i = 1 \).

**Case (2):** \( i \geq n \)

such that \( V_{(x, y)}^n(j) \rightarrow V_d \) and \( b^{n-1} \left( x_{i-1} \right) (j) \rightarrow \frac{1}{K} \). We can get

\[
\text{Limit} \left( \frac{1}{K} \left( d_{(x, y)} - \sum_{j=1}^{K} K b^{n-1} \left( x_{i-1} \right) (j) V_{(x, y)}^{n-1}(j) \right) \right) = 1
\]

because \( V_{(x, y)}^{n-1}(j) \) approaches \( V_d \) and \( b^{n-1} \left( x_{i-1} \right) (j) \) approaches \( \frac{1}{K} \). Therefore, we get the convergent condition \( 0 < \eta < \frac{2}{K^2 b^{i-1} \left( x_{i-1} \right) (j)} \) for \( i \geq n \).

**Case (3):** \( 2 \leq i \leq n-1 \)

\[
\frac{1}{K} \left( d_{(x, y)} - \sum_{j=1}^{K} K b^{i-1} \left( x_{i-1} \right) (j) V_{(x, y)}^{i-1}(j) \right) > 1/2 \text{ for } 2 \leq i \leq n-1 \text{ is true, if we let the learning constant } \eta \text{ be positive. Therefore, we get the convergent condition } 0 < \eta < \frac{4}{K^2 b^{i-1} \left( x_{i-1} \right) (j)} \text{ for } 2 \leq i \leq n-1.
\]

From the above three considered cases, we can conclude that the sufficient convergent condition of algorithm (18) is \( 0 < \eta < \frac{2}{K^2 b^{i-1} \left( x_{i-1} \right) (j)} \) for all \( i \geq 1 \).

**VI. SIMULATION RESULTS**

We take a 256×256 size image as an original image, which is shown in Fig. 5(a) for compression. The value of pixels from 0 to 1 is quantified to 256 gray levels. The signal to noise ratio (SNR) (Cierniak and Rutkowski, 2000) is defined in the Eq. (20), where RMSE presents its root mean square error after reconstruction.

\[
\text{SNR} = 20 \log \left( \frac{1}{\text{RMSE}} \right) \text{ dB} \tag{20}
\]

First, we use the one-time learning hierarchical CMAC method to compress the original image and let learning rate \( u \) be 1. Let the number of segments \( m \) equal 5, and there are 2\( m^4 \) floors in the \( i \)-th segment. The first to the fifth segment reconstructive images are shown in Fig. 5(b) to 5(e). The RMSE decreases from 0.15486 to 0.030993, and the SNR increases from 16.201dB to 30.1748dB. But bit rate (H) increases from 0.0701(b/pixel) to 0.9735(b/pixel). It
is obvious that the more segments used the less RMSE is and the higher SNR is, but the cost is the higher bit rate. We also find the boundary effect in the first three segments but not any boundary effect in last two segments because the boundary pixels have been trained.

Second, we use the memory limited CA-CMAC method to compress the original image. This method only uses one CMAC to compress image data, so no boundary effect will exist. Now, the initial learning constant $\eta$ is chosen according to the convergent condition $0<\eta<\frac{2}{K^2b(x,y)}$, and the learning constant $\eta=\frac{2}{K^2}$ is selected in the first iteration and $\eta=\frac{1}{K^2}$ is chosen in the other iteration. With one-time iteration learning reconstructive images are shown in Figs. 6(a), 6(c) and 6(e); and with 3 iterations, learning results are shown in Figs. 6(b), 6(d) and 6(f). The number of CA-CMAC floors $K$ equals eight in Figs. 6(a) and 6(b); the number of CA-CMAC floors $K$ equals four in Figs. 6(c) and 6(d); and the number of CA-CMAC floors $K$ equals two in Figs. 6(e) and 6(f). From the results of Fig. 6, it is obvious that SNR is proportional to learning iterations and bit rate per pixel, but inversely proportional to the number of floors $K$.

It is found that in the four floors memory limited CA-CMAC method the bit rate per pixel equals 0.2559(b/pixel) both appeared in one time learning in Fig. 6(c) and three times learning in Fig. 6(d).
Their RMSE and SNR are near those of the one time learning hierarchical CMAC method whose bit rate per pixel is 0.9618 (b/pixel) in Fig. 5(e) and 0.9735 (b/pixel) in Fig. 5(f). Though the reconstructive image SNR of three-exposure learning is higher than that of one time learning in memory limited CA-CMAC method as in Fig. 6, it is difficult to distinguish one time learning and three-exposure learning with the naked eye. We can conclude that the limited memory CA-CMAC compression method is better than the one-time learning hierarchical CMAC compression method. The reason is that the bit rate per pixel and computation of addition in the former method are lower than those in the latter method; however, its SNR is still higher than that in the latter method. And one-time learning in four floors limited memory CA-CMAC compression method is good enough, because it costs nearly 1/4 of the memory of original image data during compression and it needs only 4 additions per pixel during reconstruction. Besides, it is difficult to distinguish it from the original image with the naked eye.

VII. COMPARISONS

In order to compare our results with other image compression methods, we list the performance comparison of the Hierarchical CMAC (H-CMAC), the CA-CMAC, the fast discrete cosine transform (FDCT) and the Karhunen Loeve transform (KL) methods in Table 1. From this table we find that the SNR of the FDCT method is very low but it takes
more computation per pixel both in compression and reconstruction. In contrast with FDCT, the CA-CMAC method needs only \( K \) additions in a \( K \) floor CA-CMAC structure. So, of the above methods, the CA-CMAC method takes the least time to reconstruct an image. The explanations in Table 1 ** are described as follows: “The KL method is capable of considerable data compression, since its basis functions are customized for the covariance matrix of the class of image being transferred. These basis images depend upon the statistics of the image being transformed, rather than the image itself. Thus, it is reasonably likely that a group of images will have statistics similar enough that the same kernel matrix can successfully encode them. Since not all blocks of image have the same statistical character in practice, the computational load is severe for this method. Not only does the KL method become complex, also the eigenvectors of each 8\( \times \)8 matrix must be computed in advance of the transformation.”

Finally, the CA-CMAC network is applied to downsample DCT data size. It can reduce the bit rate per pixel during transmission of data and can still give high SNR after reconstruction. For comparison, we apply the CA-CMAC method and the conventional linear interpolation method to DCT compression. The sample density ratio (SR) equals sampled data size divided by original image data size. For convenience, the floor number is abbreviated to FR and learning times is abbreviated to LT. Fig. 7(a) only uses DCT and runs length encoding (RLE) to compress a 256\( \times \)256 size image data without downsampling data size. It takes 12 real multiplications and 12 real additions per pixel in an 8\( \times \)8 block DCT, and its bit rate per pixel gets \( H=0.6490 \) during transmission and gets \( SNR=34.91dB \) after reconstruction. Comparing our method CA-CMAC+ DCT+RLE in Figs. 7(d)-7(f) to the DCT+RLE with conventional linear interpolation method in Figs. 7(b)-7(c), we note that the SNR of our method is higher than that of DCT+RLE with conventional linear interpolation method at same SR. We find that if the CA-CMAC+ DCT+RLE method learns more times, it will get higher SNR without increasing sample density ratio (SR), but the cost is that more computations will be used during reconstruction. Though the SNR of reconstructed image with three-exposure learning shown in Fig. 7(f) is higher than that of one-time learning in the CA-CMAC method shown in Fig. 7(e), it is hard to distinguish the difference between one-time learning and three-exposure learning with the naked eye.

From Figs. 7(d) and 7(c), it is seen that the CA-CMAC+DCT+RLE compression method is better than the DCT+RLE with conventional linear interpolation method. Because the bit rate of the former method is lower than that of the latter one and its SNR is higher than the latter method. Meanwhile, one-time learning CA-CMAC+DCT+RLE method is good enough

<table>
<thead>
<tr>
<th>Method</th>
<th>( H )</th>
<th>Structure</th>
<th>( RMSE )</th>
<th>( SNR ) (dB)</th>
<th>Computation per pixel in the compression process</th>
<th>Computation per pixel in the reconstruction process</th>
</tr>
</thead>
<tbody>
<tr>
<td>H-CMAC</td>
<td>0.0701</td>
<td>1st segment</td>
<td>0.155</td>
<td>16.20</td>
<td>0.125add</td>
<td>16add</td>
</tr>
<tr>
<td></td>
<td>0.2020</td>
<td>2nd segment</td>
<td>0.090</td>
<td>20.91</td>
<td>0.375add</td>
<td>24add</td>
</tr>
<tr>
<td></td>
<td>0.4597</td>
<td>3rd segment</td>
<td>0.059</td>
<td>24.61</td>
<td>0.8753add</td>
<td>28add</td>
</tr>
<tr>
<td></td>
<td>0.9618</td>
<td>4th segment</td>
<td>0.032</td>
<td>30.02</td>
<td>1.875add</td>
<td>30add</td>
</tr>
<tr>
<td></td>
<td>0.9735</td>
<td>5th segment</td>
<td>0.031</td>
<td>30.17</td>
<td>1.875add</td>
<td>30add</td>
</tr>
<tr>
<td>One-time learning</td>
<td>0.1319</td>
<td>8 floors</td>
<td>0.057</td>
<td>24.79</td>
<td>6add+6mul</td>
<td>8add</td>
</tr>
<tr>
<td>CA-CMAC</td>
<td>0.2559</td>
<td>4 floors</td>
<td>0.035</td>
<td>29.05</td>
<td>4add+4mul</td>
<td>4add</td>
</tr>
<tr>
<td></td>
<td>0.5039</td>
<td>2 floors</td>
<td>0.021</td>
<td>33.45</td>
<td>2add+2mul</td>
<td>2add</td>
</tr>
<tr>
<td>3 exposure</td>
<td>0.1319</td>
<td>8 floors</td>
<td>0.053</td>
<td>25.59</td>
<td>18add+18mul</td>
<td>8add</td>
</tr>
<tr>
<td>CA-CMAC</td>
<td>0.2559</td>
<td>4 floors</td>
<td>0.032</td>
<td>30.03</td>
<td>12add+12mul</td>
<td>4add</td>
</tr>
<tr>
<td></td>
<td>0.5039</td>
<td>2 floors</td>
<td>0.015</td>
<td>36.29</td>
<td>6add+6mul</td>
<td>2add</td>
</tr>
<tr>
<td>FDCT</td>
<td>0.3792</td>
<td>8( \times )8 block</td>
<td>0.020</td>
<td>34.22</td>
<td>12add+12mul per pixel</td>
<td>12add+12mul per pixel</td>
</tr>
<tr>
<td>KL</td>
<td>**</td>
<td>low</td>
<td>high</td>
<td>Solve eigenvectors of 8( \times )8 matrix for each 8( \times )8 block</td>
<td>Multiplications of two matrices</td>
<td></td>
</tr>
</tbody>
</table>

Note: add=additions, mul=multiplications, and ** are described in the Section VII.
in Fig. 7(d). It costs near 1/36.92 memory of original image data during transmission and it takes only 1/4 computations of DCT because SR equals 1/4 during the compression and reconstruction processes. It can still get high SNR (29 dB) at the low bit rate per pixel at $H=0.2167$ after reconstruction.

**VIII. CONCLUSIONS**

Two methods are proposed to compress the transmitted data to save memory and transmitting time. The first method is the one-time learning hierarchical CMAC method and the second is the memory limited CA-CMAC method for image data compression and reconstruction. The one-time learning hierarchical CMAC method is used when a coarse image needs to be sent to the receiver initially and then the image quality is gradually improved at the request of the receiver. In other words, the receiver recognizes the image content at an early stage, and then can decide whether further transmission is necessary or not. Consequently, transmission time can be saved. When the transmitting channel data is limited, the memory limited CA-CMAC method can be used to decrease the bit rate per pixel. It not only can decrease the bit rate per pixel during the compression process but also can reduce the number of additions per pixel during the reconstruction process.
Both proposed methods unlike the conventional compression methods, use no filtering technique in either compression or reconstruction, and they have the following advantages:
(a) They do not suffer from problems of blocking effect and boundary effect.
(b) They cost very little computation both in compression and reconstruction.
(c) The coarsest reconstructed image can be quickly produced.
(d) All the reconstructed images are equal to the original image in size.
(e) One-time learning of the limited memory CA-CMAC method is good enough for compressing image data, and it has high SNR after reconstruction.

We can conclude that the limited memory CA-CMAC compression method is better than the one-time learning hierarchical CMAC compression method. The reason is that the bit rate per pixel and computation of addition in the former method are lower than those in the latter method, and its SNR is higher than that in the latter method. In addition to the above advantages, the one-time learning in four floors limited memory CA-CMAC method is good enough. Because it costs near 1/4 of the memory of the original image data during the compression process, and it needs only 4 additions per pixel during the reconstruction process; and it is also difficult to distinguish it from the original image with the naked eye.

ACKNOWLEDGMENT

This work was supported by the National Science Council of the Republic of China, under the contract NSC-90-2213-E036-010.

NOMENCLATURE

\[ b^{l}_{i,s}(j) \] the credit assigned basis function in the CA-CMAC
\[ d(x, y) \] the destined value at pixel \((x, y)\)
\[ e(x, y) \] the error between the destined value \(d(x, y)\) and output value \(g(x, y)\) of the CMAC
\[ e_{i}(s, y) \] the error between the destined value \(d(x, y)\) and output value \(g_{i}(x, y)\) in the \(i\)-th segment of the hierarchical CMAC
\[ f^{l}_{i,s}(j) \] the learned times of the \(j\)-th hypercube in the \(i\)-th iteration mapped by the pixel \((x, y)\)
\[ g(x, y) \] the output value at pixel \((x, y)\) generated from the CMAC
\[ g_{i}(x, y) \] the output value at pixel \((x, y)\) generated from the \(i\)-th segment of the hierarchical CMAC
\[ H \] the bit rate per pixel
\[ K \] the number of floors of the CMAC
\[ K_{i} \] the number of floors in the \(i\)-th segment of the hierarchical CMAC
\[ m \] the number of segments in the hierarchical CMAC
\[ N \] the number of states to be distinguished for each dimension
\[ N_{i} \] the number of hypercubes used in \(i\)-th segment of the hierarchical CMAC
\[ N_{m} \] the number of hypercubes needed to distinguish \(N^{m}\) pixels in the CMAC
\[ u \] the learning rate
\[ V(x, y)(j) \] the stored data on the \(j\)-th floor hypercube mapped by pixel \((x, y)\) in the CA-CMAC
\[ w(x, y)(j) \] the stored data on the \(j\)-th floor hypercube mapped by pixel \((x, y)\) in the CMAC
\[ \eta \] the learning constant

REFERENCES


T. Tao et al.: The One-Time Learning Hierarchical CMAC and the Memory Limited CA-CMAC for Image Data

10, pp.1393-1401.


Manuscript Received: Jan. 10, 2002
Revision Received: Jun. 02, 2002
and Accepted: Aug. 05, 2002