Proceedings of the 6th Nordic Signal Processing Symposium - NORSIG 2004
June 9 - 11, 2004

Support Vector Machines and Quad-trees Applied to Image Compression

Saavedra Ernesto[1], Grauel Adolf[1], and Morton Danny[2]

[1] University of Applied Sciences Southwestpahlia
Soest Campus
Center of Computational Intelligence & Cognitive Systems
Lübecker Ring 2, D-59494 Soest,
GERMANY
Tel: +049 2921 378450, Fax:+049 2921 378451.
Email: saavedra@fh-swf.de

[2] Bolton Institute of Higher Education
Deane Road BL3 5AB. Bolton,
ENGLAND
Tel: +44 1204 903040
Email: dm3@bolton.ac.uk

ABSTRACT

We present an approach based on Support Vector Machines **(SVM)** and quad-tree decomposition for compressing still images. Unlike **JPEG**, the method applies the discrete cosine transform **(DCT)** to regions of variable size, re-scale them to a unique block size, and before the coefficients are entropy encoded, they are approximated by a **SVM** model. The method was applied to grey scale and colour images and the obtained results improved JPEG's performance at medium and low bit rates.

Keywords: Image Compression, Support Vector Learning, JPEG, Quad-tree decomposition.

1. INTRODUCTION

Among the several image compression standards available, the Joint Photographic Experts Group **(JPEG)** is probably the most popular algorithm for still image compression. **JPEG** was approved by the International Organization of Standard (ISO) 10918 in 1994. At the time of its approval , **JPEG** gave the best compression rates (commonly smaller than 1/10) for natural images clearly outperforming other methods, and therefore it established rapidly in most all fields that require image compression. The key element in **JPEG** is the **DCT** on fixed-size regions of 8x8 pixels. Although such a size is a good compromise, it is often found in natural imagery regions bigger than 8x8 which are quite similar and that could be more effectively compressed if they were detected and properly encoded. Quad-tree decomposition [1] is an analysis technique that subdivides an image into blocks that are more homogeneous than the image itself and hence it provides information about the structure of the image. Therefore, by applying quad-tree decomposition it could then be possible to improve the performance of block-based compression algorithms like **JPEG** provided a proper encoding of the data related to each block. In this article we describe an algorithm that achieves this by restricting the size of the blocks, and that in the worst case in which the decomposition returns only regions of 8x8, its performance will be similar as **JPEG.**

Support Vector Machines are an algorithm introduced by Vapnik and co-workers [2]. They are based on the idea that if input points are mapped to a high dimensional feature space then, a separating hyper-plane can be found. **SVM** and kernel methods have been applied to wide class of problems including approximation and classification and they have proven a remarkable performance on real world problems. (Recall [3] for a introduction to **SVM** and a review on their applications). **SVM** has also been used to image compression either by applying them directly to the image surface [4] or to the transformed surface [5]. In this paper we describe an approach that applies **SVM** to the AC coefficients after quantization. This has the effect of re-scaling the data to be encoded and to keeping the sign of many coefficients resulting in an improvement of compression at medium and low bit rates.

In the next section we briefly describe the **JPEG** baseline compression algorithm and a modification of it by including quad-tree decomposition in the compression scheme and compare their performance in order to figure out the contribution of the approach to compression. We called this approach **JPEGqt**, which stands for **JPEG**-like approach based on quad-tree decomposition. In section 3, we describe the new approach which includes **SVM** before entropy encoding (**JPEGqt - SVM**), for afterwards comparing its performance with **JPEG**. The paper finishes with a discussion on the achieved results.

2. JPEG ALGORITHM

Briefly, **JPEG** applies **DCT** on each 8x8 block of the decomposed channels (three in the case of colour images, one in case of grey images), for afterwards quantizing them. Zig-zag scanning is applied to the blocks resulting in a matrix of ***m*** rows and 64 columns, where ***m*** is the number of 8x8 blocks of the channel. Differential pulse code modulation **(DPCM)** is applied on the first column (DC coefficients), and run-length encoding **(RLE)** on each row starting from column 2 to 64 (AC coefficients).

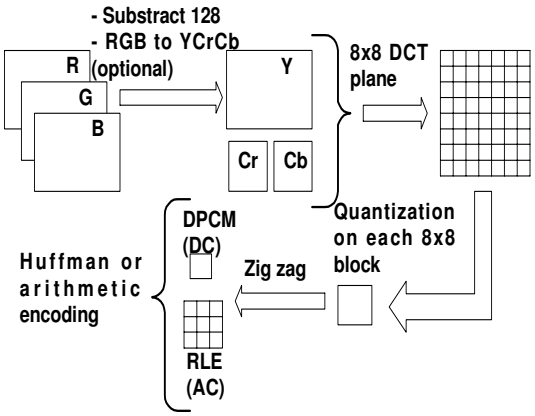

Fig.1 JPEG encoding process

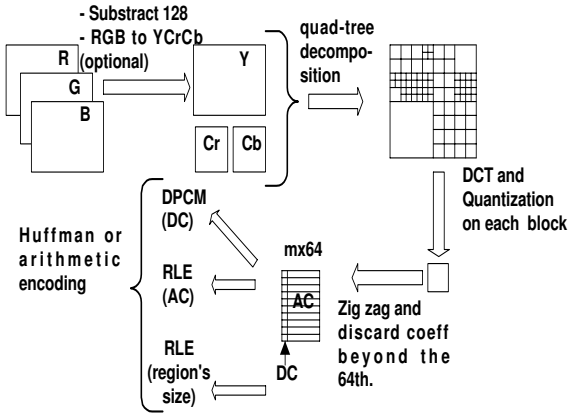

Fig.2 JPEGqt encoding process

Finally, the data is entropy encoded either by using Huffman or arithmetic encoding. (see [6] for a detailed description of the algorithm). The **JPEG** algorithm for the encoding process is depicted in Fig.1.

Although JPEG demonstrated to be an excellent algorithm for compressing natural images at low and medium bit rates, this can still be improved if the channels are decomposed in blocks of variable size but restricted to a set of sizes multiple of 8. In the following section we describe this approach

2.1 JPEGqt APPROACH

After transforming and scaling the **RGB** channels to luminance, chrominance **YcrCb,** we apply quad-tree decomposition on each of the channels but restricting the blocks size to either 8, 16, 32 and 64 pixels in order to minimize the additional data that has to be loss-less encoded. Also, by restricting the sizes to square blocks which are multiple of the smallest it will be only necessary to encode their size. This will also make possible to encode them by using a maximum of two bits. We assign the value '00' to the most frequent blocks (usually the 8x8 regions) and the maximum value '11' to the blocks which least appeared (usually the 64x64). We apply then **RLE** on each bit level and entropy-encode using Huffman encoder. In the extreme case that only smallest blocks of 8x8 are found the compression rates will be similar to **JPEG** and there will be no contribution of the method to compression. After applying **DCT** and quantization to each block of variable size, only the first 64 **DCT** coefficients of each block are taken and the rest discarded. This, based on the fact that blocks with bigger size tend to be smoother, and therefore 64 coefficients would be enough to represent the region. The approach is depicted in Fig. 2.

Fig. 3 shows the **pnsr-rate** performance for **JPEG** baseline (red), **JPEG** with specific Huffman tables (blue), **JPEGqt** restricted to blocks of 8x8 (green), and **JPEGqt** using blocks of 8, 16 32, 64 pixels (black) for the test image 'Lena.bmp'. Taking a look on this figure we see that the performance of **JPEGqt** restricted to blocks of 8x8 is even better than **JPEG** (blue). This is due to the way of scanning the data either horizontally or vertically. Our implementation scans the data vertically and for this image this resulted in a better performance. However, this is completely dependent on the image and there will be cases in which horizontal scanning performs better

3. SVM & IMAGE COMPRESSION

One of the most remarkable features of **SVM** is that the algorithm, minimizes the number of parameters required to represent a data pattern. This could be used for compressing data although [7] disagree with this for the case of high-dimensional or noise data. If we consider the problem of image compression as the problem of representing two-dimensional data with a low level of noise, it might also be possible to apply **SVM** in order to find a model which represent the data with a minimum number of parameters. On the other hand, SVM machines comprise the solution of a quadratic problem whose size is equal to the number of samples. Many approaches that deal with this problem can be found in literature, but the fact that **JPEG** and quad-tree decomposition split the data as part of their algorithms, permits to directly apply **SVM** on the **DCT** blocks. Among the several algorithms for solving **SVM**, we have implemented the Least Square **SVM (LS-SVM)** [8]. When LS-SVM are used, the found support vectors are located at the borders of the error tube and because of this it is possible to obtain **SVM** networks with less training error and same number of support vectors.

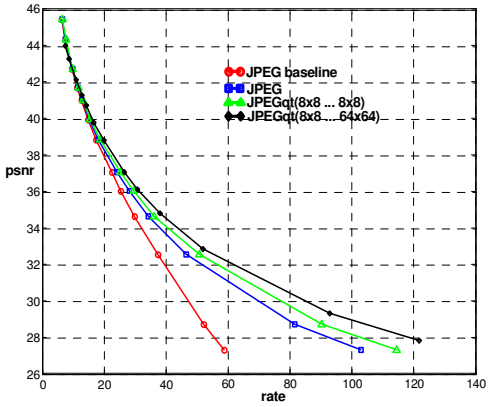

Fig.3 Comparison JPEG, JPEGqt

3.1 JPEGqt and SVM

As an aim to improve the compression rates obtained by using **JPEGqt** and following the approaches suggested in [5], we have applied **SVM** to compress the AC coefficients of the ***mx64*** matrix resulting from discarding coefficients and zig-zagging. Regarding the configuration parameters for the **SVM**, we have used a particular kernel somehow similar to the function: $f(x) = \sin(x)/x$ and to the $\delta(x)$ function with the maximum activation value centred at the position of the data (Fig.4) and programmed the insensitivity zone to 10% of the maximum absolute value of the AC coefficients at each row. By using this type of kernel, the weights of the **SVM** model become half of the size of the quantized AC coefficients and therefore it reduces the number of bits to used to encode them. Also AC coefficients which are smaller than the error tube will be set to zero, thus the **SVM** would emulate a quantizer with variable dead zone. It seems also that using a kernel with a very small negative part contribute to transport the sign of the data without using additional bits for encoding them. The approach is depicted in Fig. 5.

Fig 6. shows the **psnr-rate** curve when the method was applied to the same image as in Fig. 3 (cyan curve), using as well the same parameters for quad-tree decomposition. In order to show the effect of the negative and positive small values in the kernel, Fig.6 also includes the curves when such lateral small values were set to zero (green curve).

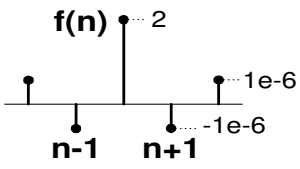

Fig. 4. SVM kernel

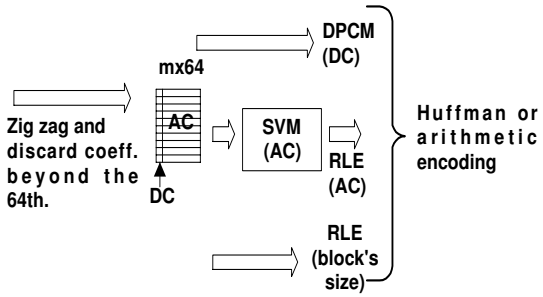

Fig.5 JPEGqt – SVM encoding process

4. DISCUSSION

We have presented two approaches for improving the performance of **DCT** block-based still image compression methods as it the case of **JPEG**. Both methods, when combined clearly outperform **JPEG** at medium and high compression rates. Unlike **JPEG**, the methods we propose split the data in regions of different size based on the criteria of similarity between neighbour pixels. This permits to apply DCT on larger and smoother regions yielding higher compression rates. The second approach uses a block based on SVM after quantization which additionally compresses the data before being entropy-encoded.

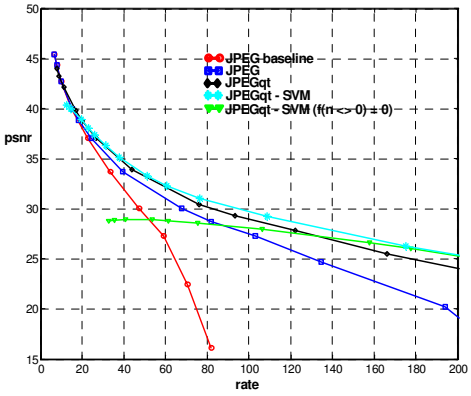

Fig.6 Comparison JPEG, JPEGqt and JPEGqt-SVM

Regarding to the usage of **SVM** for predicting AC coefficients, it turns out that at medium and high compression rates the **SVM** model is able to carry additional information by means of the small lateral values in the kernel function, which is reflected in a better quality for the same compression rate, (Fig. 6, cyan curve). JPEGqt is a fast approach that splits the image data in regions of variable size, the contribution of this method to compression is significant and its inclusion as part of the JPEG codec is feasible provided a way to transport the regions's size as part of the JPEG file, which

could be achieved by encoding them in the comments' section of the file.

ACKNOWLEDGEMENT

This work was supported by German Ministry of Research and Technology, Federal Republic of Germany **(BMBF).**

REFERENCES

1. MATLAB reference manual.
2. Vapnik V. The Nature of Statistical Learning Theory. Springer Verlag. 1995.
3. Kecman Vojislav. Learning and Soft Computing. Support Vector Machines, Neural Networks and Fuzzy Logic Models. The MIT Press, Cambridge, MA, 2001.
4. Robinson J., Kecman V. The Use of Support Vectors Machines in Image Compression. Proceedings of the EIS'2000. Paisley, Scotland, U.K. June 27 - 30. 2000.
5. Saavedra E, Grauel A, Morton, D. Combined Methods for Image Compression. pp 233-235. Recent Advances in Intelligent Systems and Signal Processing, WSEAS Conference. Corfu Island, Greece, July 2003.
6. Pennebaker William, Mitchell L. JPEG. Still image compression standard. Kluwer Academic Publishers; 1st. edition, 1993.
7. Smola Alex, Schölkopf B. A tutorial on Support Vector Regression. P 73 NeuroCOLT2. 1998.
8. Suykens J.A.K., Lukas L., Vandewalle L. Sparse Approximation Using Support Vector Machines. johan.suykens@esat.kuleuven.ac.be