Enlarging Context with Low Cost: Efficient Arithmetic Coding with Trimmed Convolution

Anonymous ECCV submission

Paper ID 340

Abstract. Arithmetic coding is an essential class of coding techniques which have been widely used in various data compression systems and exhibited promising performance. One key issue of arithmetic encoding method is to predict the probability of the current coding symbol from its context, i.e., the preceding encoded symbols, which usually can be executed by building a look-up table (LUT). However, the complexity of LUT increases exponentially with the length of context. Thus, such solutions are limited in modeling large context, which inevitably restricts the compression performance. Several recent deep neural network-based solutions have been developed to account for large context, but are still costly in computation. The inefficiency of the existing methods are mainly attributed to that probability prediction is performed independently for the neighboring symbols, which actually can be efficiently conducted by shared computation. To this end, we propose a trimmed convolutional network for arithmetic encoding (TCAE) to model large context while maintaining computational efficiency. As for trimmed convolution, the convolutional kernels are specially trimmed to respect the compression order and context dependency of the input symbols. Benefited from trimmed convolution, the probability prediction of all symbols can be efficiently performed in one single forward pass via a fully convolutional network. Furthermore, to speed up the decoding process, a slope TCAE model is presented to divide the codes from a 3D code map into several blocks and remove the dependency between the codes inner one block for parallel decoding, which can 60× speed up the decoding process. Experiments show that our TCAE and slope TCAE attain better compression ratio in lossless grayscale image compression, and can be adopted in CNN-based lossy image compression to achieve state-of-the-art rate-distortion performance with real time encoding speed.

Keywords: Image compression, arithmetic encoding, trimmed convolution, convolutional network

1 Introduction

Image compression aims to encode image with less bits, and can provide an effective solution to save storage requirement and transmission bandwidth. Based on whether distortion is allowed in the reconstructed image or not, image compression can be classified into lossy and lossless methods. For both the two categories

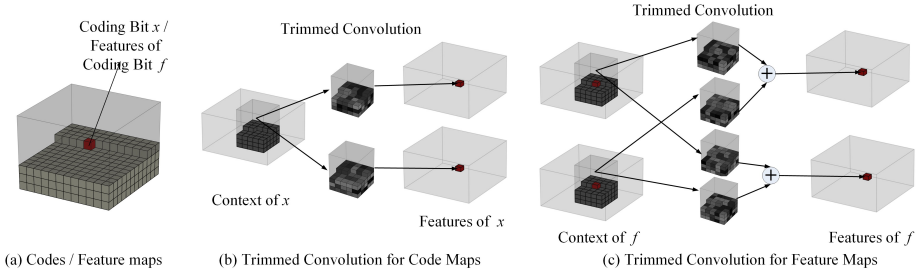


Fig. 1: Illustration of the trimmed convolution operations used in the input layer and the hidden layers. The red blocks represent the current code or feature to be encoded. The blank area in the code maps or feature maps represent the non-encoded codes or features along with the encoding order.

of approaches, lossless entropy encoding is an important building block for generating the compressed representation.

One key issue of entropy encoding is to predict the probability of the current symbol to be encoded. In some earlier approaches, such as JPEG, the frequency of the symbols is directly counted and the classical Huffman coding approach is used to compress the codes. While, recent approaches tend to take benefit from the image context information for better predicting the probability of symbols. In JPEG 2000, the EBCOT coder [1] is employed to model the context and approximate the probability, and the Binary Arithmetic Coding-MQ-Coder is utilized to compress the codes. In H.264/AVC, the context-adaptive binary arithmetic coding (CABAC) [2] is introduced to model the context of the preceding two encoded symbols for compressing the codes with arithmetic coding. CABAC adopts look-up table (LUT) to predict the probability of the current symbol based on its context information.

The PixelCNN and PixelRNN models have validated the effectiveness of deep neural networks (DNNs) for capturing highly complex and long-range dependence between pixels. However, since the un-decoded bits can not be utilized for probability prediction in the decoding process, standard DNNs can not be directly applied for entropy coding in image compression. Toderici et al. [3] present a recurrent neural network (RNN) based model, where a binary RNN is trained to estimate the probability of bits for better arithmetic coding. However, their model needs to conduct one forward pass to predict each bit, making the RNN based encoder [3] computationally expensive. Recently, Li et al. [4] replace the unavailable values in the context cuboid with pre-defined values and train a shallow CNN to predict the probability of current bit. Even though the CNN solution [4] speeds up the RNN based encoder, it also performs probability prediction independently for the neighboring symbols and remains inefficient.

In this paper, we propose a trimmed convolutional network for arithmetic encoding (TCAE) which can greatly speed up the process of arithmetic encod-

ing. Specifically, TCAE introduces a new class of trimmed convolution operations to accelerate the probability prediction step in the entropy coding process. Compared with the standard convolution operations which incorporate all the surrounding information to generate the output, trimmed convolution utilizes a binary mask to avoid the adopting of certain input values. Equipped with trimmed convolution kernels, the proposed TCAE approach is able to avoid the involvement of non-encoded symbols for probability prediction. Consequently, a fully convolutional network architecture can be directly adopted for probability prediction, making our TCAE highly efficient for arithmetic encoding.

In Fig. 1 (a) and (c), we illustrate the application of trimmed convolution kernels on the the input layer and hidden feature maps, respectively. For the current bit to be encoded in the input layer (red pixel in Fig. 1 (a)), it should be excluded from the context for probability prediction. While, for the value in the same location of hidden feature map, as it only conveys the information from the corresponding context area, we should include it in the coding process. We will give more details on mask settings for sophisticated multiple filters and 3D data in Section 3.

Furthermore, to accelerate the decoding speed, we introduce a slope TCAE schedule, which divides the codes of 3D code maps into several blocks by requiring that the codes inner one block are independent. Therefore, we can decode the codes in one block simultaneously, while the code blocks still need to be decoded in order. Concretely, (i, j, k) denotes the coordinate of the 3D code map $\mathbf{x}$. The t-th code block is defined as $CB_t(\mathbf{x}) = \{x_{i,j,k} | i + j + k = t\}$.

In this paper, we evaluate our TCAE and slope TCAE on both the image lossless and lossy compression tasks. For the lossless compression task, we adopt the proposed algorithms to compress the grayscale images, and achieve higher compression ratio than the existing lossless compression methods, such as PNG, JPEG-LS and JPEG2000-LS. While, for the lossy compression task, TCAE and slope TCAE is adopted to compress the intermediate codes of the lossy compression system. We take a recently proposed CNN-based system [4] as an example, and utilize our methods to compress the binary codes and importance map generated by the encoder. Compared with [4], our TCAE can not only improve the compression performance due to the consideration of large context, but also be significantly faster in terms of encoding speed. Benefited from trimmed convolution, the compression system [4] with slope TCAE can encode the image in real time, and 60× speed up the decoding process.

To sum up, the contribution of this work is four-fold:

- Trimmed convolution is incorporated with fully convolutional network to perform probability prediction to all bits within one single forward pass.
- A trimmed convolutional arithmetic encoder is developed to encode either grayscale image or intermediate codes of CNN-based lossy compression system.
- A slope TACE is introduced to break down the dependency of a block of codes for parallel decoding which 60× speeds up the decoding process.

– Experiments show that for lossless grayscale image compression TCAE can achieve higher compression ratio than PNG and JPEG2000-LS. Based on the CNN-based lossy image compression system [4], TCAE can achieve real time encoding speed without sacrifice of the rate-distortion performance.

2 Related Work

2.1 Lossless image compression standards

Lossless image compression has been investigated for decades to compress image into a smaller size without losing any information. Most existing methods share similar pipeline, i.e., building a statistical model and then using it to guide the mapping of input data stream to bit sequences by encoding the high frequency symbol with fewer bits than low frequency symbol. JPEG-LS [5] takes use of the LOCO-I algorithm which adopts the median edge detection predictor to predict the image and then compress the residual with the context-based entropy coding. JPEG2000-LS, the lossless version of JPEG 2000 standard [6], is based on reversible integer wavelet transform (biorthogonal 3/5). PNG [7] first uses a lossless filter to transform the images and then compresses the transformed data with the DEFLATE algorithm which is a combination of LZ77 and Huffman coding. Both TIFF and GIF apply Lemple-Ziv-Welch (LZW) algorithm [8] which is a dictionary based method for lossless image compression.

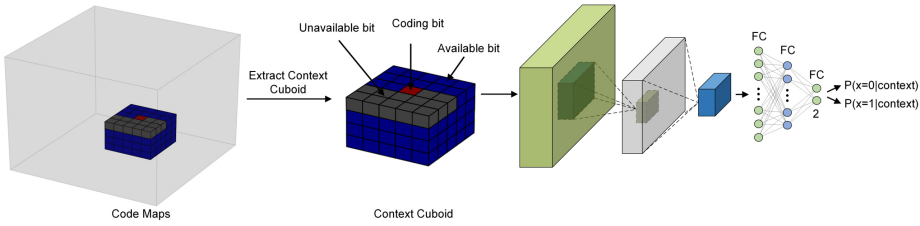
2.2 Entropy encoding

Entropy encoding is a lossless data compression scheme which has played an essential role in both lossy and lossless image compression systems. Run length encoding, Huffman coding [9], Golomb-Rice coding [10] and arithmetic coding [11] are several representative entropy encoding techniques. Run length coding is a very simple lossless data compression scheme, where a consecutive symbol sequence are stored as the symbol and its count.

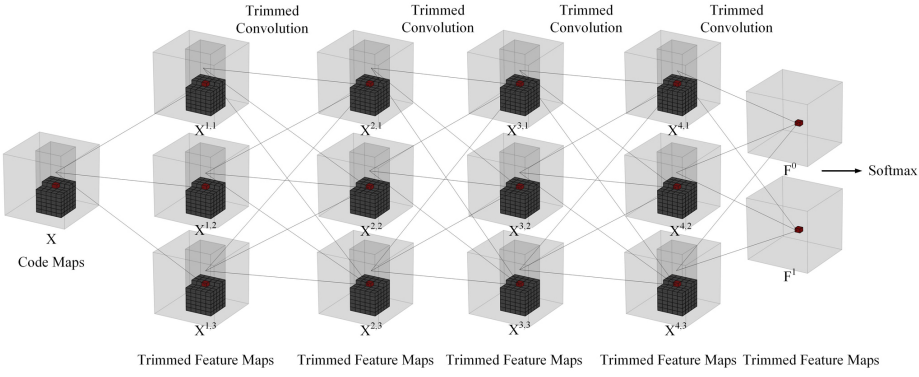
Huffman coding [9] is a variable-length coding scheme that encodes high frequency symbols with short codeword and low frequency symbols with long codeword. In Golomb-Rice code [10], the input value is first divided into two parts, i.e., quotient and remainder, and truncated binary encoding is then adopted to encode the remainder. Arithmetic coding [11] first predicts the probability of the current symbol to be encoded and then uses it to divide the current interval into sub-intervals for encoding the updated sequence.

2.3 Deep networks based entropy encoding

PixelCNN [12] and PixelRNN [13] have shown great power in image generation and modeling highly complex and long-range dependence. However, both of them are designed for 2D image and only model the context on the 2D plane instead of 3D code maps. Toderici et al. [14] learn a binary recurrent network to model



(a) Framework of traditional convolutional network for entropy probability estimator used in [4]. One bit is processed as a sample.



(b) Framework of trimmed convolutional network for entropy probability estimator. The whole code maps are processed as a sample.

Fig. 2: Comparison between trimmed convolutional networks and traditional convolutional networks for entropy probability estimator.

the context both from the same coding plane and the codes from the previous coding planes. However, the probability of each bit should be estimated with one forward pass, making binary RNN estimator computational very inefficient. Li et al. [4] introduce a convolutional entropy estimator to model the code context for the 3D code maps by extracting a specially defined context cube and processing the cube with a small convolutional network. Compared with binary RNN estimator, the convolutional estimator is relatively more efficient, but repeating calculation remains inevitable due to the overlap of the context of overbearing bits. Actually, the inefficiency of the methods in [14,4] is mainly attributed to that probability prediction is performed independently for the neighboring symbols. In this work, we introduce a class of trimmed convolution to make the probability prediction more efficient by sharing computation. By stacking multiple trimmed convolution layers, our TCAE can also exploit large context to improve compression performance.

3 Trimmed Convolutional Arithmetic Encoding

In this section, we present our trimmed convolutional arithmetic encoding model (TCAE). As illustrated in Fig. 2b, TCAE utilizes the trimmed convolutional

networks to predict the probability of codes from their context in one single forward pass. Before introducing the trimmed convolutional network, we first describe arithmetic coding and coding context in Sections 3.1 and 3.2. Then trimmed convolutional network is provided for context modeling and probability prediction in Section 3.3, and a variant of TCAE, the slope TCAE is proposed for parallel decoding. The model objective is described in Section 3.4.

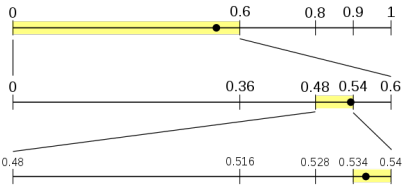
3.1 Arithmetic coding

Arithmetic encoding is an entropy encoding scheme for lossless data compression. Given a string of symbols, entropy encoding assigns fewer bits for frequently occurring symbols and more bits for not-so-frequently occurring ones. Different from other entropy encoding algorithms such as Huffman coding, arithmetic coding encodes the entire string of symbols into a single number in the interval of $[0, 1]$. Denote by k the number of symbols occurred in the coding system. Given a new symbol x_i and the current interval, arithmetic coding first predicts the probabilities $p(x_i = t|x_{i-1}, \ldots, x_0)$ that the new symbol belongs to each value t, and the current interval is further divided according to the predicted probabilities. In order to encode the updated sequence, the current interval is then updated based on the ground truth of the new symbol as well as its predicted probability. For example, Fig. 3a illustrates the arithmetic coding result of a sequence (0, 2, 3) for a coding system with the symbols (0, 1, 2, 3) and the discrete distribution with probabilities (0.6, 0.2, 0.1, 0.1).

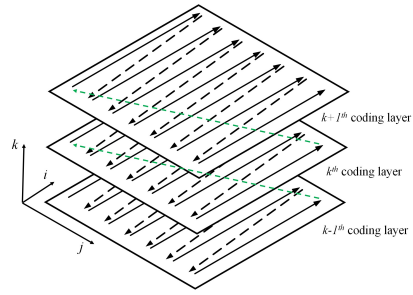
3.2 Coding schedule and context of 3D cuboid

In this work, we focus on the arithmetic encoding of 3D binary code cuboid $\mathbf{X} = \{x_{i,j,k}|0 \leq i \leq W-1, 0 \leq j \leq H-1, 0 \leq k \leq C-1\}$. As illustrated in Fig. 3b, beginning at $x_{0,0,0}$, we follow the following order to encode $\mathbf{X}$: (i) $x_{i+1,j,k}$ is encoded after $x_{i,j,k}$ until $i = W-1$; (ii) when $i = W-1$, $x_{0,j+1,k}$ is encoded after $x_{i,j,k}$ until $j = H-1$; (iii) when $i = W-1$ and $j = H-1$, $x_{0,0,k+1}$ is encoded after $x_{i,j,k}$.

One key issue of arithmetic encoding is to predict the probability of a symbol from its context, i.e., the preceding encoded symbols. Given the position (p, q, r) of 3D cuboid, we denote its full context as $CTX(x_{p,q,r})$. Taking the encoding order of $\mathbf{X}$ into account, $CTX(x_{p,q,r})$ can be defined as $CTX(x_{p,q,r}) = \{x_{i,j,k}|\{k < r\} \vee \{k = r, j < q\} \vee \{k = r, j = q, i < p\}\}$. Unfortunately, the length of the full context $CTX(x_{p,q,r})$ is not fixed and varies by the position (p, q, r), making it difficult to learn probability prediction model on $CTX(x_{p,q,r})$. Note that the context close to the bit to be encoded plays more important role in probability prediction. Thus we adopt a fixed length context defined as $CTX_f(x_{p,q,r}) = \{x_{i,j,k}|\{r - c \leq k < r, |i - p| \leq w, |j - q| \leq h\} \vee \{k = r, q - h \leq j < q, |i - p| \leq w\} \vee \{k = r, j = q, p - w \leq i < p\}\}$. It is natural to expect that large context (i.e., large w, h, c) benefits probability prediction, and in the following we suggest to use trimmed convolutional networks for improved and efficient arithmetic encoding.



(a) Arithmetic encoding of a sequence (0,2,3) for a coding system with the symbols (0,1,2,3) and the i.i.d distribution with probabilities (0.6,0.2,0.1,0.1)



(b) Coding schedule for binary code cuboid.

Fig. 3: Arithmetic encoding and coding schedule.

3.3 Trimmed convolutional networks

With the fixed length context defined above, we can extract a $(2w+1)\times(2h+1)\times(c+1)$ cuboid $\mathbf{x}_{p,q,r} = \{x_{i,j,k} | \{r-c \leq k \leq r, p-w \leq i \leq p+w, q-h \leq j \leq q+h\}$. As shown in Fig. 2a, the blue voxels represent the encoded bits before $x_{p,q,r}$, the red voxel denotes the next bit to be encoded, and the gray ones are the other non-encoded bits. Then, for each location $x_{p,q,r}$, Li et al. [4] adopt a CNN architecture of three convolution layers followed by three fully connected layers to predict $x_{p,q,r}$ from its context cuboid $\mathbf{x}_{p,q,r}$.

One main obstacle, which restricts the shared computation of the probability prediction, is that the introduction of non-encoded bits (i.e., the red and gray voxels in Fig. 2a). In [4], Li et al. suggest to assign a special default value to the current non-encoded bits in the context cuboid $\mathbf{x}_{p,q,r}$. For example, when predicting $x_{p,q,r}$, $x_{p,q,r}$ should be replaced with default value [4]. When encoding the next bit after $x_{p,q,r}$, the original value of $x_{p,q,r}$ should be used in context modeling. Therefore, fully convolutional network cannot be directly utilized for context modeling, and the probability prediction is performed independently for each $x_{p,q,r}$, leading to repeating computation and encoding inefficiency [4].

Fortunately, the value of each location only have two choices, i.e., the original value $x_{p,q,r}$ and the default value for non-encoded bit. Moreover, given the voxel $x_{p,q,r}$ to encode, the positions of all non-encoded bits are also fixed. Without loss of generality, we let the default value be 0. Thus, we can introduce a group of trimmed convolution operators to incorporate with fully convolutional network for context modeling. Denote by $\mathbf{w}^0$ a 3D convolution kernel $\mathbf{w}^0 = \{w_{t,i,j,k} | -w_0 \leq i \leq w_0, -h_0 \leq j \leq h_0, -t \leq k \leq C-t-1\}$. Then the convolution result $(\mathbf{X} * \mathbf{w}^0)$ at the location (p,q,r) can be written as,

$$(\mathbf{X} * \mathbf{w}^0)(p,q,r) = \sum_{l-i=p, m-j=q, n-k=r, t=r} x_{l,m,n} w^0_{t,i,j,k}. \quad (1)$$

However, both the context and non-encoded bits have effect on the convolution result $(\mathbf{X} * \mathbf{w}^0)(p,q,r)$. We note that the positions of non-encoded bits are fixed

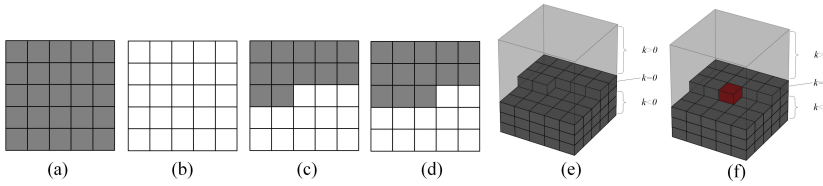


Fig. 4: Mask planes with respect to k for trimmed convolution kernels with the size of 5×5. The gray value and red value denotes 1 and the white value denotes 0. (a) $k < 0$, (b) $k > 0$, (c) $k = 0$ for the input layer, (d) $k = 0$ for the hidden layers, (e) 3D kernel mask for input layer, (f) 3D kernel mask for hidden layer.

and pre-defined with respect to $\mathbf{w}^0$, and thus can introduce a mask $\mathbf{m}$ of $\{0, 1\}$ to exclude them in convolution. To this end, $m_{i,j,k}$ is defined as 1 if $x_{p+i,q+j,r+k}$ is encoded before $x_{p,q,r}$, and 0 otherwise. The trimmed convolution is thus defined as,

$$\mathbf{X}^1 = \mathbf{X} * (\mathbf{m} \circ \mathbf{w}^0), \tag{2}$$

where $*$ denotes the convolution operator, and $\circ$ denotes the element-wise multiplication operator. With trimmed convolution, we can safely avoid the effect of non-encoded bits in context modeling while maintaining the efficiency of fully convolutional network for predicting probabilities of all bits in one forward pass.

In the following, we first use single convolution kernel as an example to explain the settings of $\mathbf{m}$, which are different for the input layer and the hidden layers. For the input layer, when predicting the probability of $x_{p,q,r}$, both $x_{p,q,r}$ and the bits encoded after $x_{p,q,r}$ should be masked out in trimmed convolution. Following the definition of context in Section 3.2, we define the mask $\mathbf{m}^0$ for the input layer as,

$$m^0_{ijk} = \begin{cases} 1, \text{if}(k<0) \vee (k=0, j<0) \vee (k=0, j=0, i<0) \\ 0, \text{otherwise} \end{cases} \tag{3}$$

When it comes to the hidden layer $\mathbf{X}^d$ $(d \geq 1)$, we note that $x^d_{p,q,r}$ only conveys the context information of $x_{p,q,r}$ and should not be excluded in the further context modeling. Therefore, we modify the definition of the mask $\mathbf{m}^d$ $(d \geq 1)$ for hidden layer as,

$$m^d_{ijk} = \begin{cases} 1, \text{if}(k<0) \vee (k=0, j<0) \vee (k=0, j=0, i\leq 0) \\ 0, \text{otherwise}. \end{cases} \tag{4}$$

Using a 3D convolution kernel with the size of $5 \times 5 \times C$ as an example, Fig. 4 illustrates the representative mask planes with respect to k. As shown in Fig. 4ab, when $k < 0$ $(k > 0)$, the k-th mask plane is a matrix of 1s (0s) for both the input layer and the hidden layers. when $k = 0$, the center position is masked out in the mask plane for the input layer (see Fig. 4c) but are included for the hidden layers (see Fig. 4d).

The trimmed convolution in Eqn. (2) only uses one set of convolution kernels in each layer, which is limited in probability prediction. Thus, we extend the trimmed convolution to the multiple convolution kernel form. Suppose there are g_{in} groups of feature maps $\mathcal{X}^d = \{\mathbf{X}^{d,1}, ..., \mathbf{X}^{d,g_{in}}\}$ in the d-th layer and g_{out} groups of feature maps $\mathcal{X}^{d+1} = \{\mathbf{X}^{d+1,1}, ..., \mathbf{X}^{d+1,g_{out}}\}$ in the $(d+1)$-th layer. Each group of feature map has the same size with the input cuboid $\mathbf{X}$. The group trimmed convolution is then defined as,

$$\mathbf{X}^{d+1,g'} = \sum_{g=1}^{g_{in}} \mathbf{X}^{d,g} * (\mathbf{m}^d \circ \mathbf{w}^{d,g,g'}), \tag{5}$$

where $\mathbf{X}^{d,g}$ denotes the g-th group of feature map in $\mathcal{X}^d$, $\mathbf{X}^{d+1,g'}$ denotes the g'-th group of feature map in $\mathcal{X}^{d+1}$, $\mathbf{m}^d$ is the mask for the d-th layer, and $\mathbf{w}^{d,g,g'}$ is the convolution kernel to connect $\mathbf{X}^{d,g}$ and $\mathbf{X}^{d+1,g'}$.

3.4 Slope TCAE

With the proposed TCAE, the possibility of all the codes can be predicted simultaneously with the trimmed convolutional network in coding period. However, in the decoding stage, the prediction of the possibility of x_i, $p(x_i|x_{i-1}, \ldots, x_0)$, depends on $x_{i-1}, \ldots, x_0$, making that x_i should be decoded after $x_{i-1}, \ldots, x_0$. Thus, the possibility prediction of codes still need to be processed in serial order. This becomes a bottleneck of the deep learning based context modeler which is usually speed up with GPU and parallel computation.

To speed up the decoding process of TCAE, we should break down some dependency among codes. That is to say, for a code x_i, it does not depend on all the $x_{i-1}, \ldots, x_0$ but only part of them. By dividing the codes $\mathbf{x}$ into different blocks and supposing the codes inner block are independent, we can break down the dependency of the codes inner block and parallel predict the codes in one block simultaneously. Here, the t-th code block is defined as:

$$CB_t(\mathbf{x}) = \{x_{i,j,k} |\ i + j + k = t\}. \tag{6}$$

Then, the possibility of x_i can be modeled as $p(x_i|\ CB_{t-1}(\mathbf{x}), \ldots, CB_0(\mathbf{x}))$ where $x_i \in CB_t(\mathbf{x})$. Thus, all the x_i in the code block $CB_t(\mathbf{x})$ share the same context and can be predicted in the same time. Since the codes in one block exactly fall on a slope plane of a cuboid, we call the parallel context modeler as slope TCAE.

With the slope TCAE, the context should be modified. The context for code $x_{p,q,r}$, $CTX(x_{p,q,r}) = \{x_{i,j,k} |\ i + j + k < p + q + r\}$. For feature context, all the features in the plane $i + j + k = p + q + r$ are predicted with the context $CTX(x_{p,q,r})$ and can be further used as the context of the feature $f_{p,q,r}$ to predict $x_{p,q,r}$, $CTX_f(f_{p,q,r}) = \{f_{i,j,k} |\ i + j + k \leq p + q + r\}$.

The coding schedule is refined as coding the 3D codes map according to the defined blocks, from $CB_0(\mathbf{x})$ to $CB_{H+W+C}(\mathbf{x})$. Inner one block $CB_t(\mathbf{x})$, we sort the elements in ascending order first with the index k then with the index i. The

new schedule is used for slope TCAE. Further more, the mask $\mathbf{m}^0$ and $\mathbf{m}^d$ are modified as:

$$m^0_{ijk} = \begin{cases} 1, \text{if } \ i+j+k < 0 \\ 0, \text{otherwise} \end{cases} \tag{7}$$

$$m^d_{ijk} = \begin{cases} 1, \text{if } \ i+j+k \leq 0 \\ 0, \text{otherwise.} \end{cases} \tag{8}$$

3.5 Model objective and learning

Given all the model parameters $\mathcal{W} = \{\mathbf{w}^{d,g,g'}\}$, the output of the trimmed convolutional network can be written as $F(\mathbf{X};\mathcal{W})$. $F(\mathbf{X};\mathcal{W})$ includes m parts, $(F(\mathbf{X};\mathcal{W}))^t_{p,q,v}$ denotes the predicted probability of $x_{p,q,v} = t$ with $t = 0, \ldots, m-1$. And m is the number of different code value in the input code maps. We adopt the length of the codes after arithmetic encoding as the model objective,

$$\ell(\mathcal{W};\mathbf{X}) = \sum_{p,q,v} \sum_{t=0}^{m-1} -s(x_{p,q,v},t) \log_2 \left(F(\mathbf{X};\mathcal{W})\right)^t_{p,q,v} \tag{9}$$

where $s(x_{p,q,v},t) = 1$ when $x_{p,q,v} = t$, and $s(x_{p,q,v},t) = 0$ otherwise. According to the Shannon's theorem, the compression ratio is defined as the ratio between uncompressed size and compressed size, and can be written as

$$\begin{aligned} \frac{1}{r(e)} = \frac{\ell(\mathcal{W};\mathbf{X})}{CHW \log_2 m} = \frac{1}{\log_2 m}[\frac{1}{CHW} \\ \sum_{p,q,v} \sum_{t=0}^{m-1} -s(x_{p,q,v},t) \log_2 \left(F(\mathbf{X};\mathcal{W})\right)^t_{p,q,v}]. \end{aligned} \tag{10}$$

Thus, it is reasonable to use the objective in Eqn. (9) to learn probability prediction model for entropy encoding.

By minimizing the model objective defined in Eqn. (9), the trimmed convolutional network $F(\mathbf{X};\mathcal{W})$ can be learned from training data in an end-to-end manner. In this work, we adopt the ADAM solver [15] to learn the model parameters. The model is trained with the learning rate of 3×10^{-4}, 1×10^{-4}, 3.33×10^{-5} and 1.11×10^{-5}. The smaller learning rate is adopted until the objective with the larger one stops decreasing.

4 Experiments

Three groups of experiments are conducted to test the proposed trimmed convolutional arithmetic encoding (TCAE) and slope TCAE. The first is the lossless compression of grayscale image. To satisfy the requirement of 3D binary cuboid, we take the 8-bit representation of grayscale image as the input. To verify the context modeling ability of TCAE, we further use the trained grayscale image

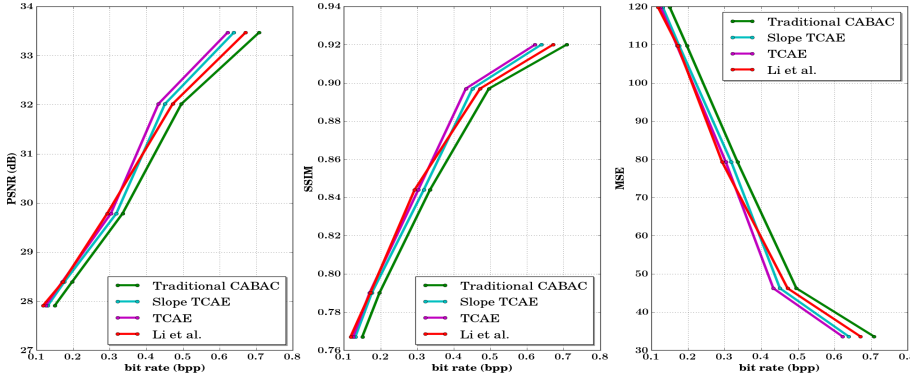


Fig. 5: The effect of TCAE and slope TCAE on the rate-distortion performance of the lossy image compression system [4]

Table 1: Architecture of trimmed convolutional networks.

Layer	Activation size
Input	$h \times w \times c$
$g \times c \times 5 \times 5 \times c$ trimmed conv, pad 2, stride 1	$g \times h \times w \times c$
$g \times c \times 5 \times 5 \times c$ trimmed conv, pad 2, stride 1	$g \times h \times w \times c$
Trimmed residual block, $g \times c$ 3D filters	$g \times h \times w \times c$
Trimmed residual block, $g \times c$ 3D filters	$g \times h \times w \times c$
Trimmed residual block, $g \times c$ 3D filters	$g \times h \times w \times c$
Trimmed residual block, $g \times c$ 3D filters	$g \times h \times w \times c$
$m \times c \times 5 \times 5 \times c$ trimmed conv, pad 2, stride 1	$m \times h \times w \times c$

predictor for image inpainting. The third group of experiments is the incorporation with CNN-based lossy compression. We use TCAE to compress the binary codes and importance map generated by [4], and compare the rate-distortion performance with the baseline method [4], JPEG, JPEG-2000, and Ballé et al. [16].

4.1 Network architecture and parameter setting

Table 1 gives a general network structure for the TCAC. Let the size of the input 3D code map is $w \times h \times c$. g is the number of groups used in each convolution layer. m is the number of labels to be predict. The output represents the possibility the code belongs to each class, i.e. $(F(\mathbf{X}; \mathcal{W}))^t_{p,q,v}$, where p, q, v is the index of the code in the 3D code map $\mathbf{X}$ and t is the index of each class with $t = 0, \ldots, m-1$. In TCAE, we also introduce the trimmed residual block, which consists of two trimmed convolution layers with each followed by the ReLU nonlinearity. The skip connection is also added from the input to the output of the residual block. For defining the model objective in Eqn 9, feature map rearrangement is adopted to reshape the input and output of the trimmed convolutional network.

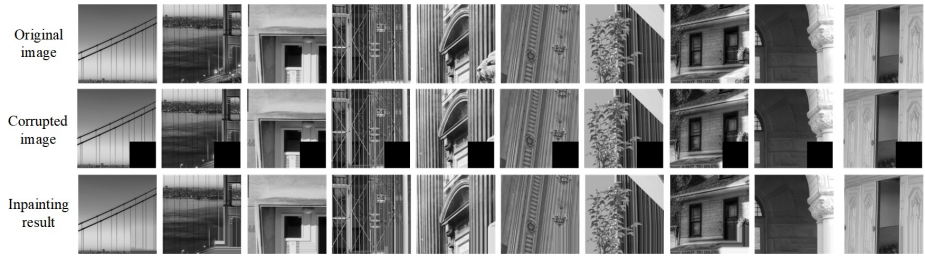


Fig. 6: Inpainting results of TCAE.

Concretely, the input is reshaped into a one dimensional vector with the size of $chw \times 1$, while the output is reshaped into a 2D matrix with the size of $chw \times m$.

All the networks are trained on 10,000 high quality images from the ImageNet [17] and tested on the Kodak PhotoCD image dataset. Due to that TCAE is a fully convolutional network, it can be trained and tested using images with any size. In the experiments, we set the image size as 128×128 for training lossless grayscale image compression modeler. Note that the input of TCAE is required to be 3D binary cuboid. Thus, we transform the discrete grayscale image to its 8-bit representation, i.e. 8 binary bit planes, resulting in a $128 \times 128 \times 8$ 3D binary cuboid.

4.2 Lossless grayscale image compression

For lossless grayscale image compression, our TCAE is simply deployed to the 8-bit representation of grayscale image without any transform. We compare TCAE with several popular lossless compression standards, including GIF, TIFF, PNG, JPEG-LS and JPEG2000-LS. For PNG, JPEG2000-LS, JPEG-LS, GIF and TIFF, all the compression ratio is calculated with the results generated with the Matlab2015b. The average compression ratio on 24 images from Kodak dataset is given in Table 2. Our TCAE achieves the best compression ratio (2.00), which is much better than the second best method (1.79 for JPEG2000-LS).

Table 2: Results of lossless grayscale image compression.

Method	TIFF	GIF	PNG	JPEG-LS	JPEG2000-LS	TCAE	slope TCAE
Compression Ratio	1.01	1.13	1.61	1.57	1.79	2.00	1.99

4.3 Image inpainting

The trained lossless image compression model can also be adopted for the task of image inpainting. We corrupt a rectangle area with the size of $\frac{1}{9}$ from the right bottom of the original image and then fill in the blank area with the TCAE context predictor. We just assign 0 or 1 to each binary plan of the grayscale image in the blank area with the probability predicted by the TCAE. Figure 6 gives the inpainting results of TCAE, which shows that TCAE can achieve plausible results in recovering the image textures and structures.

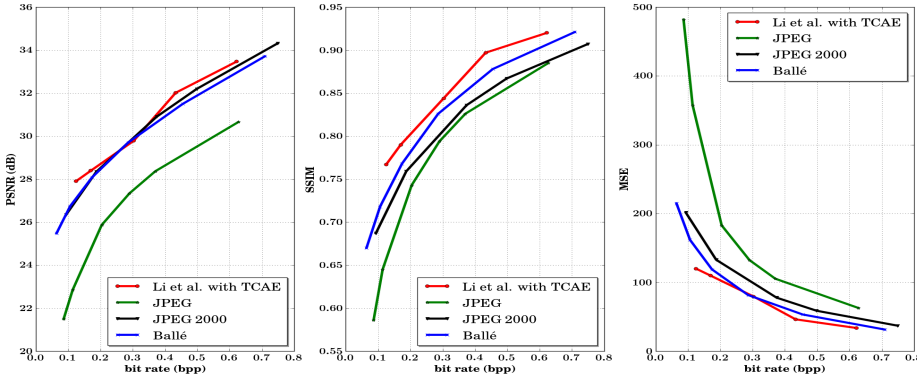


Fig. 7: Comparision of rate-distortion performance of the TCAE extention of [4] and other lossy image compression methods.

4.4 CNN-based lossy image compression [4]

Entropy encoding can also be used in the CNN-based lossy image compression system to compress the intermediate codes. In our experiment, we take the system [4] as an example, and replace the convolutional entropy encoder in [4] with our TCAE. Fig. 5 shows the rate-distortion curves obtained using our TCAE and convolutional entropy encoder [4]. Here, SSIM, MSE, and PSNR are adopted as the distortion performance metrics, and bit per pixel (*bpp*) is used as the indicator of compression rate. As shown in Fig. 5, our TCAE and slope TCAE achieves comparable performance with convolutional entropy encoder [4]. And all the convolutional network based context models, e.g., TCAE, slope TCAE and the convolutional entropy encoder, can get better compression ratio than the traditional CABAC context modeler. When $bpp \geq 0.4$, TCAE and slope TCAE performs better than convolutional entropy encoder [4], which can be attributed to the ability of TCAE in modeling large context. Fig. 7 further compares the rate-distortion curves obtained using Li et al. [4] with TCAE, JPEG, JPEG 2000, and Ballé [16]. When $bpp \geq 0.4$, Li et al. [4] with TCAE outperforms the competing methods, regardless of any distortion metrics. The result indicates that TCAE can be incorporated with the CNN-based lossy image compression to boost rate-distortion performance.

The most prominent merit of TCAE is its efficiency in encoding due to the incorporation of trimmed convolution and fully convolutional network. To illustrate this, Table 3 reports the run time for encoding the intermediate codes [4]. Here, Groups 1∼5 represent the five settings of model parameters [4] to obtain the compression models at different *bpp*s in the range of $[0.118, 0.671]$. All the experiments are conducted on a computer with the GTX TitanX GPU of 12GB memory. The reported time is based on the encoding of the importance map and binary codes of a $752 \times 496 \times 3$ image, and we do not include the time for generating the intermediate codes. In general, it takes about 0.051 s and 0.098 s to generate the intermediate codes with 64 and 128 channels, respectively. From Table 3, it can be seen that our TCAE is 7∼20 times faster than the entropy en-

Table 3: Run time (in seconds, s) used for encoding the intermediate codes in [4].

Group	Convolutional entropy encoder [4]			TCAE		
	Binary	Importance	Total	Binary	Importance	Total
Group 1	0.181	0.092	0.274	0.035	0.003	0.038
Group 2	0.249	0.092	0.342	0.035	0.003	0.038
Group 3	0.468	0.092	0.561	0.035	0.003	0.038
Group 4	0.710	0.092	0.802	0.035	0.003	0.038
Group 5	1.199	0.115	1.313	0.161	0.005	0.166

coder used in [4]. With the introduction of trimmed convolution, our TCAE can be performed at a near real time speed (e.g., 25 fps) to encode the intermediate codes of images with the size of $752 \times 496 \times 3$.

4.5 Slope TCAE vs. TCAE

As shown in Table 2 and Figure 5, slope TCAE only has small drop both in grayscale image compression and compressing 3D code maps due to removing the dependency among codes inner code blocks. It verifies our assumption, the codes inner one block are independent, is reasonable. For the encoding efficiency, the encoding speed of slope TCAE is the same as TCAE. For the decoding efficiency, with the small drop of the compression ratio, the slope TCAE get a huge improvement in decoding efficiency.

In decoding period, the convolutional entropy encoder [4] uses about 212.32 seconds to decompress the 3D binary code map with the size of $94 \times 68 \times 64$. Cropping the 3D binary codes into smaller size can speed up the decoding speed. With experiments, by cropping the 3D maps into $16 \times 16 \times 64$, the compression ratio has no clear drop. With the strategy, it takes 15.64 seconds to decode one image for convolutional entropy encoder [4] by parallel decoding several small 3D code maps. With the same setting, TCAE takes 42.31 seconds to decode the 3D code blocks. However, the slope TCAE only needs 0.62 seconds for decoding the 3D code maps, which is more than $60\times$ faster than TCAE and 25 times faster than convolutional entropy encoder [4].

5 Conclusion

The paper presents a trimmed convolutional network for arithmetic encoding (i.e., TCAE) of 3D binary cuboid. Benefitted from trimmed convolution, we can utilize the fully convolutional network for performing probability prediction to all bits in one single forward pass, making it feasible to model large context while maintaining computational efficiency. To speed up the decoding process, we introduce a slope TCAE schedule to divide the 3D code maps into different blocks and remove dependency among codes inner blocks for parallel decoding. In comparison with traditional methods such as PNG and JPEG2000-LS, our TCAE and slope TCAE achieve better compression ratio for lossless grayscale image compression. Moreover, it can also be incorporated with the CNN-based lossy image compression system (e.g., [4]) to compress the intermediate codes, and exhibits its superiority in large context modeling and real time encoding speed.

References

1. Medouakh, S., Baarir, Z.: Entropy encoding ebcot (embedded block coding with optimized truncation in jpeg2000). IJCSI **8** (2011)
2. Marpe, D., Schwarz, H., Wiegand, T.: Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard. IEEE Transactions on circuits and systems for video technology **13**(7) (2003) 620–636
3. Toderici, G., O'Malley, S.M., Hwang, S.J., Vincent, D., Minnen, D., Baluja, S., Covell, M., Sukthankar, R.: Variable rate image compression with recurrent neural networks. arXiv preprint arXiv:1511.06085 (2015)
4. Li, M., Zuo, W., Gu, S., Zhao, D., Zhang, D.: Learning convolutional networks for content-weighted image compression. arXiv preprint arXiv:1703.10553 (2017)
5. Weinberger, M.J., Seroussi, G., Sapiro, G.: The loco-i lossless image compression algorithm: Principles and standardization into jpeg-ls. IEEE Transactions on Image processing **9**(8) (2000) 1309–1324
6. Skodras, A., Christopoulos, C., Ebrahimi, T.: The jpeg 2000 still image compression standard. IEEE Signal processing magazine **18**(5) (2001) 36–58
7. Boutell, T.: Png (portable network graphics) specification version 1.0. (1997)
8. Welch, T.A.: A technique for high-performance data compression. Computer **6**(17) (1984) 8–19
9. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the IRE **40**(9) (1952) 1098–1101
10. Gallager, R., Van Voorhis, D.: Optimal source codes for geometrically distributed integer alphabets (corresp.). IEEE Transactions on Information theory **21**(2) (1975) 228–230
11. Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Communications of the ACM **30**(6) (1987) 520–540
12. van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al.: Conditional image generation with pixelcnn decoders. In: Advances in Neural Information Processing Systems. (2016) 4790–4798
13. Oord, A.v.d., Kalchbrenner, N., Kavukcuoglu, K.: Pixel recurrent neural networks. arXiv preprint arXiv:1601.06759 (2016)
14. Toderici, G., Vincent, D., Johnston, N., Hwang, S.J., Minnen, D., Shor, J., Covell, M.: Full resolution image compression with recurrent neural networks. arXiv preprint arXiv:1608.05148 (2016)
15. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
16. Ballé, J., Laparra, V., Simoncelli, E.P.: End-to-end optimized image compression. arXiv preprint arXiv:1611.01704 (2016)
17. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE (2009) 248–255