DCT-Conv: Coding filters in convolutional networks with Discrete Cosine Transform

Karol Chęciński, Paweł Wawrzyński
Institute of Computer Science
Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warsaw, Poland
karol.checinski.stud@pw.edu.pl, pawel.wawrzynski@pw.edu.pl

***Abstract*—Convolutional neural networks are based on a huge number of trained weights. Consequently, they are often data-greedy, sensitive to overtraining, and learn slowly. We follow the line of research in which filters of convolutional neural layers are determined on the basis of a smaller number of trained parameters. In this paper, the trained parameters define a frequency spectrum which is transformed into convolutional filters with Inverse Discrete Cosine Transform (IDCT, the same is applied in decompression from JPEG). We analyze how switching off selected components of the spectra, thereby reducing the number of trained weights of the network, affects its performance. Our experiments show that coding the filters with trained DCT parameters leads to improvement over traditional convolution. Also, the performance of the networks modified this way decreases very slowly with the increasing extent of switching off these parameters. In some experiments, a good performance is observed when even 99.9% of these parameters are switched off.**

***Index Terms*—neural networks, parameters reduction, convolution, discrete cosine transform**

I. Introduction

Convolutional neural networks are now the most efficient tool for image analysis tasks such as face recognition [1], medical diagnostic [2, 3] or objects detection [4]. Contemporary models such as VGG [5], ResNet [6], GPipe [7] have up to hundreds of millions of trained parameters. Training them requires huge volumes of data or it is prone to overtraining and poor behavior on unseen images.

A way to avoid the aforementioned unfortunate alternative is as follows: For each filter (or each filter's part) in the network there exists a vector of trained parameters; its dimension is smaller than the size of the filter (or its part); a filter (or its part) is produced from the trained parameters with a certain fixed transformation. Fewer trained parameters represent a similar diversity of filters. The above fixed transformation is a key element that needs to be designed for this approach to be successful.

In this study we analyze the aforementioned transformation based on (Inverse) Discrete Cosine Transform, (I)DCT. DCT can be applied to a matrix $m \times n$ of real numbers to produce a matrix of the same size of real numbers — DCT coefficients. The matrix of DCT coefficients can be transformed back, with IDCT to the original matrix. For the original matrix which is a part of a real image, often some DCT coefficients are close to zero. Therefore, JPEG compression is based on the DCT of parts of the image with skipping DCT coefficients that are close to zero.

The contribution of this paper can be summarized in the following points:

- We propose to train DCT coefficients of filters in convolutional neural networks rather than the filters themselves.
- We demonstrate on four large learning problems that the above modification causes a significant improvement in the performance of the networks on the test data.
- We analyze how the performance of the above networks is influenced by switching off the DCT coefficients of the filters, i.e. setting them equal to zero and excluding them from training. We demonstrate that the performance decreases very slowly with increasing the extent of the switching off.

The remainder of the paper is organized as follows. The next section presents related work. For this paper to be self-contained, Sec. III presents DCT, IDCT, and some of their properties. Sec. IV presents the DCT-Conv layer, a layer in which filters are defined by trained DCT coefficients. Sec. V presents an empirical study. The last section concludes the paper.

II. Related work

This paper is related to two issues elaborated in the literature: The first one is how to reduce the number of trained parameters in neural networks. The second one is the application of the DCT in the processing of data by neural networks.

A. Reduction of trained parameters

There are two main motives to reduce the number of trained parameters in a neural network. The first one is a better generalization, thus a lower demand for data. The second one is a lower demand for memory to store those parameters which enables broader applicability, e.g. in mobile devices and embedded systems.

In [8] a method of pruning parameters of small absolute value was introduced that enabled reducing the number of trained parameters from AlexNet by 85% without accuracy deterioration. In [9], 75% of the parameters were reduced in a convolutional network due to the decomposition of the filter matrix. In [10] hashing functions were applied to divide parameters into bins in which they are trained together. [11] shows that a significant number of parameters can be reduced by the appropriate shaping of the input and a network structure.

An approach to parameters' reduction in convolutional networks is the application of the same filters in different scales [12] and rotated by different angles [13]. This way a network does not need to learn multiple filters to represent the same pattern at different scales and angles. Consequently, different filters may cover better all the patterns present in the data that the network needs to be able to identify.

B. Application of Discrete Cosine Transform in neural networks

DCT has been applied to process input images for convolutional neural networks. In [14] this approach was combined with the transformation of images to YCbCr color space and applied to image classification and verified on MNIST and CIFAR-10 dataset. Also, images transformed to the frequency domain are analyzed in specific applications. E.g. in [15], features of images are produced by convolutional networks processing both original images and their frequency representations. That architecture was applied to the classification of cells suspected of acute lymphoblastic leukemia (ALL). DCT was applied to compress filters in convolutional networks in [16].

III. Discrete Cosine Transform

Let $x \in \mathcal{R}^{m\times n}$ be a matrix with entries $x_{i,j}, i = 0,...,m-1, j = 0,...,n-1$. DCT-II (Discrete Cosine Transform type II)[1] of x produces a matrix, $X \in \mathcal{R}^{m\times n}$, whose entries, DCT coefficients, $X_{k,l}, k = 0,...,m-1, l = 0,...,n-1$ are given by the formula:

$$X_{k,l} = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} x_{i,j} \cos\left[\frac{\pi}{n}\left(j+\frac{1}{2}\right)l\right] \times \times \cos\left[\frac{\pi}{m}\left(i+\frac{1}{2}\right)k\right]. \tag{1}$$

Given DCT coefficients X, the original image x can be reconstructed with IDCT-II (Inverse Discrete Cosine Transform type II) as follows

$$x_{i,j} = \sum_{k=0}^{m-1}\sum_{l=0}^{n-1} X_{k,l} \cos\left[\frac{\pi}{n}\left(j+\frac{1}{2}\right)l\right] \times \times \cos\left[\frac{\pi}{m}\left(i+\frac{1}{2}\right)k\right]. \tag{2}$$

The formulae for DCT-II and IDCT-II are thus very similar. They apply to matrices. Formulae for DCT-II and IDCT-II for vectors are easily obtained by setting above $n = 1$. Formulae for DCT-II and IDCT-II for tensors of higher order are analogical to those above.

[1]There are several types of DCT; type II is the most popular one in multimedia.

In words, the DCT coefficient $X_{k,l}$ says how strong in x the component is whose frequency over the first coordinate is proportional to k, and whose frequency over the second coordinate is proportional to l. In particular, $X_{0,0}$ is the sum of all elements in x.

DCT is applied in JPEG compression: An image is divided into squares and their DCT is computed. Only DCT coefficients with sufficiently large absolute values are put into the compressed file. Usually, that means that high frequency components that correspond to noise and small details in the image, insignificant for the viewer anyway, are skipped.

IV. Method

The general idea analyzed in this paper is as follows. We take a convolutional layer of a neural network. Whenever the filters in the layer are in use, they result from IDCT on trained weights, instead of being trained parameters themselves. Specifically, tensors with trained weights are transformed into filters of the same shape through IDCT. However, some of the weights are set to zero. Hence, there are fewer trained parameters than in the original convolutional layer, but the filters within the layer are still diverse enough to respond to different patterns in data.

A. DCT-Conv layer

Trained parameters (weights) of a DCT-Conv layer create a 4th-order tensor in $\mathcal{R}^{N\times C\times H\times W}$, where N is a number of filters, C is a number of channels, and H, W are the height and width of a filter, respectively. A filter is a sequence of C slices – $H \times W$ matrices. IDCT is performed on the weights tensor to create a tensor of filters of the same shape, such that IDCT is performed independently for each slice. Once created, the tensor of filters is used as usual in a convolutional layer.

In some architectures, like ResNet, there are filters with $W = H = 1$. Effectively, a layer of such filters is defined by a matrix in $\mathcal{R}^{N\times C}$ (tensor of order 2). While this matrix could also be computed on the basis of its trained DCT coefficients, this is problematic because of two reasons. Firstly, while one may expect a certain regularity in relation between values in filters and their spacial coordinates, existence of a relation between values in filters and indices of channel and especially filter is less obvious. Secondly, usually $N, C \gg W, H$ which makes IDCT of the aforementioned matrix expensive. Because of these problems, in the experiments discussed below a matrix of 1×1 filters is produced with IDCT separately for its subrows of length 16.

In order to run any gradient-based learning algorithm with a network that contains a DCT-Conv layer, derivatives of the cost function need to be computed with respect to the trained weights. Let us denote the cost function by J, and suppose that its derivatives with respect to filter components, denoted by $x_{i,j}$ (2), namely $\mathrm{d}J/\mathrm{d}x_{i,j}$, are known. Then, the derivatives of the cost function with respect to the trained weights, denoted

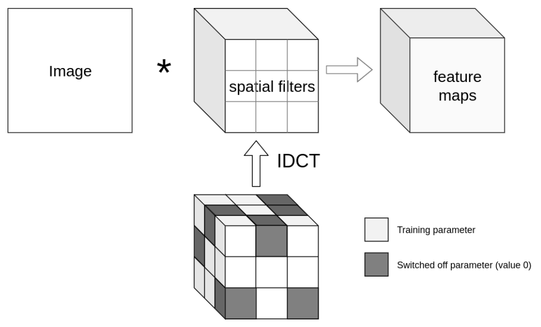

Figure 1. DCT-Conv layer. Gray weights are switched off (set to zero).

by $X_{k,l}$, are determined through error backpropagation i.e., by analyzing how $X_{k,l}$ influences different $x_{i,j}$ in (2), namely

$$\frac{\mathrm{d}J}{\mathrm{d}X_{k,l}} = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} \frac{\mathrm{d}J}{\mathrm{d}x_{i,j}} \cos\left[\frac{\pi}{n}\left(j+\frac{1}{2}\right)l\right] \times \times \cos\left[\frac{\pi}{m}\left(i+\frac{1}{2}\right)k\right]. \tag{3}$$

Some of the weights are switched off i.e., set to zero and excluded from training. They are randomly, independently between one another selected before the training. The switch-off probability is defined by a coefficient, $p \in [0, 1]$.

While in the future we plan to research more efficient schemes of switching the weights off, now we only analyze the above simple scheme. The DCT-conv layer is presented in Fig. 1.

V. Experiments

In the empirical study below we compare the performance of original networks with convolutional layers against similar architectures with DCT-Conv layers with the same number of filters of the same size. Also, we analyze how the switch-off probability impacts the performance of the networks with DCT-Conv layers. We analyze 4 benchmark learning problems, all based on the CIFAR-100 dataset of images.

A. Dataset: CIFAR-100

The CIFAR-100 dataset [17] contains 60,000 images in RGB with a size of 32x32px. Each image is of one of 100 equivocal disjoint classes. The dataset is divided into a training set (50,000 images) and a test set (10,000 images).

In our experiments the images have been normalized: For each channel, c, an average, $av(I_{tr}[\cdot,\cdot,c])$, and standard deviation, $\sigma(I_{tr}[\cdot,\cdot,c])$ were computed for the images in the training set. The normalized images feeding the neural networks are computed according to

$$\hat{I}[i,j,c] = (I[i,j,c] - av(I_{tr}[\cdot,\cdot,c])/\sigma(I_{tr}[\cdot,\cdot,c]) \tag{4}$$

where (i, j) are the coordinates.

Table I
ResNet50 architecture. ConvBlock2D and IdentBlock2D are presented in Fig. 2. The numbers in the Size column next to blocks denote the numbers of filters within convolutional layers in a block. Column 3 and 4: number of parameters in 3×3 and 1×1 filters, respectively.

ResNet50			
Layer/Block	Size	Parameters	
Conv2D	64x3x3	1792	
MaxPool2D 2x2			
ConvBl2D	64, 64, 256	36,928	37,440
IdentBl2D x2	64, 64, 256	73,856	66,256
ConvBl2D	128, 128, 512	147,584	230,528
IdentBl2D x3	128, 128, 512	442,752	395,136
ConvBl2D	256, 256, 1024	590,080	919,808
IdentBl2D x5	256, 256, 1024	2,950,400	2,627,840
ConvBl2D	512, 512, 2048	2,359,808	3,674,624
IdentBl2D x2	512, 512, 2048	4,719,616	4,199,424
GlobalAveragePooling2D			
Dense	100	204,900	

B. Problem 1. ResNet50 classifier

a) Architecture: We adopt the ResNet50 architecture with bottleneck blocks [6]. ResNets are generally deep modular convolutional networks parameterized by the number of layers. Characteristic modules in ResNets are residual blocks that combine convolution with parallel passing of the block input to its output. These by-passes prevent exploding and vanishing gradients, and enable very deep architectures.

The orthogonal initialization [18] has been applied to all the filters within the network. The output dense layer has been initialized by means of Glorot's method [19].

The network architecture is depicted in detail in Tab. I. The network has 23,676,990 trained parameters, including 11,317,248 in 3x3 filters, and 12,130,384 in 1x1 filters.

b) Training: Classic Momentum [20] was applied with a momentum decay factor of 0.9. The step-size was evolving in the training time according to the following scheme: 0.001 in epoch 1, 0.1 till epoch 60, 0.02 till epoch 120, 0.004 till epoch 160, and 0.0008 till the final epoch 200. The minibatches of size 128 were used.

C. Problem 2. VGG-16 classifier

a) Architecture: The original VGG architecture [5] was designed to process large images. The architecture used here is based on VGG-CIFAR [22]. It was optimized to process smaller images. It has a different number of dense layers, their sizes are different, and it contains batch normalization layers. Originally, VGG contains two dense layers with 4096 neurons and 1000 output neurons (the original VGG had been trained on ImageNet). VGG-CIFAR contains only one dense layer with 512 neurons and 100 output neurons (the number of classes in CIFAR-100). Additionally, as compared to the architecture from [22], we resigned from drop-out after convolutional layers. That modification leads to a noticeable improvement in performance on the test set.

The activation function in each convolutional layer is ReLU. After each convolutional and dense layer there is a batch

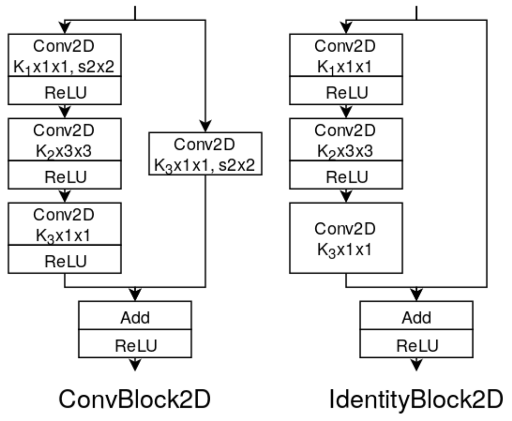

Figure 2. Structure of ConvBlock2D and IdentBlock2D. The term $K_1\times3\times3$ denotes that a convolutional layer includes K_1 filters of size 3×3. The term 2×2 denotes strides in both coordinates. The default strides are 1×1. A batch normalization block [21] is after each convolutional layer.

Table II
VGG-16 ARCHITECTURE. 64X3X3 DENOTES 64 FILTERS WITH SLICES OF SIZE 3X3. CONV2D X2 DENOTES 2 LAYERS ONE AFTER ANOTHER.

VGG-16		
Layer/Block	Size	Parameters
Conv2D x2	64x3x3	38,720
MaxPool2D 2x2		
Conv2D x2	128x3x3	221,440
MaxPool2D 2x2		
Conv2D x3	256x3x3	1,475,328
MaxPool2D 2x2		
Conv2D x3	512x3x3	5,899,776
MaxPool2D 2x2		
Conv2D x3	512x3x3	7,079,424
MaxPool2D 2x2		
Dense, ReLU	512	262,656
Dropout 0.5		
Dense, softmax	100	51,300

normalization layer. A detailed description of the architecture is presented in Tab. II. The network has 15,028,644 trained parameters, including 14,710,464 in convolutional layers.

b) Training: The network was trained with the use of the NAG algorithm. The momentum decay factor was set to 0.9. The step-size was evolving according to the formula

$$\beta_t = \beta_0(0.5^{\lfloor t/20 \rfloor}), \tag{5}$$

where t is the epoch index, $\beta_0 = 0.1$ is the initial step-size. The whole training lasted for 200 epochs. The loss function was categorical cross-entropy with L2 regularization of weights (with the coefficient of 0.0005). Minibatches of size 128 were used for the training.

D. Problem 3. Autoencoder 1

a) Architecture: The first autoencoder is taken from [23]. It contains 6 layers: 2 convolutional, 2 dense, and 2 layers with transposed convolution. Its architecture is presented in detail in Tab. III. The network has 4,204,435 trained parameters, including 5,472 in convolutional layers and those with transposed convolution.

Table III
AUTOENCODER 1. 16X3X3 DENOTES 16 FILTERS WITH SLICES OF SIZE 3X3. CONV2DT DENOTES TRANSPOSED CONVOLUTION.

Autoencoder 1		
Layer	Size	Parameters
Conv2D, ReLU	16x3x3	448
MaxPool2D 2x2		
Conv2D, ReLU	16x3x3	2,320
Dense	512	2,097,664
Dense, sigmoid	4096	2,101,248
Conv2DT, ReLU	16x3x3	2,320
UpScale2D 2x2		
Conv2DTranspose, sigmoid	16x3x3	435

Table IV
AUTOENCODER 2. 16X3X3 DENOTES 16 FILTERS WITH SLICES OF SIZE 3X3. CONV2DT DENOTES TRANSPOSED CONVOLUTION.

Autoencoder 2		
Layer	Size	Parameters
Conv2D, ReLU	16x3x3	448
Conv2D, ReLU	32x3x3	4,640
Conv2D, ReLU	64x3x3	18,496
MaxPool2D 2x2		
Conv2D, ReLU	128x3x3	73,856
Conv2D, ReLU	64x3x3	73,792
MaxPool2D 2x2		
Conv2D, ReLU	32x3x3	18,464
Conv2D, ReLU	16x3x3	4,624
Conv2D, sigmoid	8x3x3	1,160
Conv2DT, ReLU	16x3x3	1,168
Conv2DT, ReLU	32x3x3	4,640
UpSampling2D 2x2		
Conv2DT, ReLU	64x3x3	18,496
Conv2DT, ReLU	128x3x3	73,856
UpSampling2D 2x2		
Conv2DT, ReLU	64x3x3	73,792
Conv2DT, ReLU	32x3x3	18,464
Conv2DT, ReLU	16x3x3	4,624
Conv2DT, sigmoid	3x3x3	435

b) Training: The network was trained with the ADAM algorithm using the momentum decay factor of 0.9 and the step-size of 0.001. The minibatches size was 64. The loss function was the binary cross-entropy.

E. Problem 4. Autoencoder 2

a) Architecture: The second autoencoder is based on https://github.com/Puayny/Autoencoder-image-similarity. However, the convolutions in its decoder were replaced with a transposed convolution. The architecture is presented in detail in Tab. IV. The network contains 390,955 trained parameters, including 390,240 in (transposed) convolution filters.

b) Training: The network was trained with the ADAM algorithm using a momentum decay factor of 0.9 and the step-size of 0.0005 (this was the largest value that assured stable learning of the network). The minibatches size was 64. The loss function was the binary cross-entropy.

F. Implementation and computational efficiency

Our experimental software has been written in Python/Tensorflow 2.0. DCT-Conv layers were implemented

Table V
RESULTS. EACH REPORTED RESULT IS AN AVERAGE REGISTERED ON THE TEST SET AFTER TRAINING. THE AVERAGE IS OVER 5 RUNS. LEFT-MOST COLUMNS CONTAIN RESULTS FOR NETWORKS WITH CNN LAYERS. OTHER COLUMNS CONTAINS RESULTS FOR NETWORKS WITH DCT-CONV LAYERS WITH DIFFERENT PROBABILITY OF SWITCHING OFF WEIGHTS.

Problem 1, ResNet50, all filters. Result reported: Accuracy

CNN	$p = 0$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.9$	$p = 0.97$
0.6698	0.7127	0.7198	0.6845	0.7044	0.7067	0.6529

Problem 1, ResNet50, only 3x3 filters. Result reported: Accuracy

CNN	$p = 0$	$p = 0.99$	$p = 0.999$	$p = 0.9999$	$p = 1$
0.6698	0.6826	0.7331	0.7141	0.6415	0.3416

Problem 2, VGG-16. Result reported: Accuracy

CNN	$p = 0$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.9$
0.7177	0.7221	0.7088	0.7099	0.6930	0.6517

Problem 3, Autoencoder 1. Result reported: MSE for pixels in $[0, 1]$

CNN	$p = 0$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.9$
0.00136	0.00126	0.00132	0.00138	0.00151	0.00283

Problem 4, Autoencoder 2. Result reported: MSE for pixels in $[0, 1]$

CNN	$p = 0$	$p = 0.3$	$p = 0.5$	$p = 0.7$	$p = 0.9$
0.00153	0.00118	0.00135	0.00149	0.00188	0.00381

such that an additional DCT block was applied to produce filters of ordinary convolutional (or transposed convolution) layers. Switching off was implemented such that the weights were multiplied elementwise by mask tensors with appropriate number of entries set to zero.

The real time overhead resulting from adding the DCT block to the convolutional layer is comparable to the computing time that the original layer takes. Therefore, our experiments with DCT-Conv networks lasted up to twice longer than with the original convolutional networks. This time could be significantly reduced if DCT-Conv layer was implemented as a single block.

G. Results

All the experiments were performed according to the following pattern:

1) The original neural network is trained.
2) The convolutional layers of the network are replaced with DCT-Conv layers. Its trained DCT coefficients are initialized as the components of the filters were originally initialized. The network is retrained from the beginning according to the original regime.
3) The DCT coefficients of the DCT-Conv layers are reinitialized, and some of them are switched off, i.e. set equal to zero and excluded from the training. They are switched off on random, each with probability p. Technically, they are assigned random real numbers, sorted according to those numbers, and first $p100\%$ of them are switched off. Once again, the network is retrained from the beginning according to the original regime.

Note that the network architecture, as well as its training regime, has been optimized with respect to its performance when it is used without any further modifications. We do not change any of these when replacing the original convolutional layers with DCT-Convs.

The results are presented in Tab. V. Each second row in Tab. V presents performance on the test set. Each number averages 5 runs. For problems 1 and 2, the performance is expressed with accuracy. For problems 3 and 4, the performance is expressed with a mean square error. Every left column shows the performance of the original convolutional neural network. Other columns demonstrate the performance of the network with its convolutional layers replaced by DCT-Convs and their DCT coefficients switched off with a different probability, p.

a) Problem 1: Here we can observe that the network with our proposed layers outperforms the original one if only $p \leq 0.9$.

A debatable issue for this problem is how to treat 1×1 convolutions. At first, we joined them over 16 consecutive channels and performed IDCT on such packs. In another experiment we leave the original convolutional layers with 1×1 filters, and the convolutional layers with $W \times H$ filters for $W, H > 1$ are replaced with DCT-Convs. The result is stunning: Even if DCT coefficients of such layers are switched off with the probability $p = 0.999$, the network performs better than the original one. Let us consider a filter with 3×3 slices and 128 input channels. It has $3 \cdot 3 \cdot 128 = 1152$ DCT coefficients. When they are switched off with the probability $p = 0.999$, some filters are switched off entirely, and each of the others has only a few slices that are in fact operational. This happens to be enough for a fairly good performance of the network. However, it is not true that 3×3 filters are useless in this network. When they are all switched off ($p = 1$), the network performs poorly.

b) Problem 2: Here we can observe that the network with our proposed layers outperforms the original one if only p is very close to zero. However, the accuracy on the test set decreases rather slowly with growing p.

c) Problem 3: Here we can observe that the network with our proposed layers outperforms the original one if only $p < 0.5$.

d) Problem 4: Here we can observe that the network with our proposed layers outperforms the original one if only $p \leq 0.5$.

H. Summary

In all problems analyzed in the above empirical study, a network with our proposed DCT-Conv layers performed better than the original network with convolutional layers. Switching DCT coefficients off in DCT-Conv layers led to the deterioration of performance. However, the performance decreased rather slowly with the growing probability of switching off. If less than about 50% DCT coefficients were switched off, the network still performed better than the original one. However, in the case of the ResNet50 classifier (Problem 1) 90% DCT coefficients could be switched off without deterioration of performance. If only convolutional layers with 3×3 filters in ResNet50 were replaced with DCT-Convs, then 99.9% of

DCT coefficients could be switched off without deterioration of performance.

VI. Conclusions and future work

In this paper DCT-Conv layer was introduced — a layer in which trained weights are DCT coefficients of spatial filters. The DCT-Conv layer realizes the idea that a sufficiently rich set of spacial filters can have sparse frequency representation.

In four experiments with benchmark convolutional neural networks, it was demonstrated that the networks with their convolutional layers replaced by DCT-Conv layers outperform the original networks even if large part, about 50%, of their DCT coefficients are switched off (set equal to zero and excluded from training). In some cases that part could be significantly larger.

In this paper we considered switching off the DCT coefficients of the DCT-Conv layer on random with equal probability for each layer. We plan to investigate strategies of determining those probabilities for different layers and their specific DCT coefficients. Also, optimization of the DCT-Conv layers shape is a curious topic of future research.

Acknowledgment

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

References

[1] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “Deepface: Closing the gap to human-level performance in face verification,” in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 1701–1708.

[2] A. Esteva, B. Kuprel, R. Novoa, J. Ko, S. Swetter, H. Blau, and S. Thrun, “Dermatologist-level classification of skin cancer with deep neural networks,” *Nature*, vol. 542, 01 2017.

[3] R. Ghosh, K. Ghosh, and S. Maitra, “Automatic detection and classification of diabetic retinopathy stages using cnn,” in *2017 4th International Conference on Signal Processing and Integrated Networks (SPIN)*, Feb 2017, pp. 550–554.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection.” CVPR, 06 2016, pp. 779–788.

[5] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” arXiv:1409.1556, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” arXiv:1512.03385, 2015.

[7] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, “Gpipe: Efficient training of giant neural networks using pipeline parallelism,” arXiv:1811.06965, 2018.

[8] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” arXiv:1510.00149, 2015.

[9] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, “Predicting parameters in deep learning,” in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, 2013, pp. 2148–2156.

[10] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, “Compressing neural networks with the hashing trick,” in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 2285–2294.

[11] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International Conference on Machine Learning*, 2019.

[12] Y. Xu, T. Xiao, J. Zhang, K. Yang, and Z. Zhang, “Scale-invariant convolutional neural networks,” arXiv:1411.6369, 2014.

[13] D. Marcos, M. Volpi, N. Komodakis, and D. Tuia, “Rotation equivariant vector field networks,” in *The IEEE International Conference on Computer Vision*, 2017.

[14] M. Ulicny and R. Dahyot, “On using cnn with dct based image data,” in *19th Irish Machine Vision and Image Processing conference*, 2017, pp. 44–51.

[15] S. Kant, P. Kumar, A. Gupta, and R. Gupta, “Leukonet: Dct-based cnn architecture for the classification of normal versus leukemic blasts in b-all cancer,” arXiv:1810.07961, 2018.

[16] Y. Wang, C. Xu, S. You, and D. Tao, “Cnnpack: Packing convolutional neural networks in the frequency domain,” in *Advances in Neural Information Processing Systems 29*, 2016.

[17] A. Krizhevsky, “Learning multiple layers of features from tiny images,” University of Toronto, Tech. Rep., 05 2012.

[18] A. Saxe, J. McClelland, and S. Ganguli, “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks,” arXiv:1312.6120, 2014.

[19] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[20] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999.

[21] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *32nd International Conference on Machine Learning*, 2015.

[22] S. Liu and W. Deng, “Very deep convolutional neural network based image classification using small training sample size,” in *3rd IAPR Asian Conference on Pattern Recognition*, 2015, pp. 730–734.

[23] P. Wawrzyński, “Efficient on-line learning with diagonal approximation of loss function hessian,” in *International Joint Conference on Neural Networks*, 2019.