Factorized Convolution Kernels in Image Processing

Alexander W. Bergman, David B. Lindell
Stanford University
450 Serra Mall, Stanford, CA 94305
awb@stanford.edu

Abstract

This report describes the theory, implementation and performance of convolutional neural networks with factorized convolutional kernels on a denoising task. Two kernel factorization methods are compared in terms of accuracy in denoising, number of parameters required, and number of multiplication and addition operations in evaluation.

1. Introduction

Convolutional neural networks (CNNs) are a class of deep neural networks which have enjoyed success in learning tasks related to image analysis. Specific uses of CNNs in imaging include applications in LiDAR systems like object detection [5] and helping perform image processing pipeline steps such as denoising [6]. However, with the increasing resolution of images and the increasing complexity of CNN models, the memory and computational costs of storing and using CNNs quickly can become intractable. For example, applications of CNNs implemented on embedded systems or systems with constrained hardware have limited computing and memory resources. Many computational imaging systems share this requirement, so it is of interest that image processing tasks implemented with CNNs also be computationally and memory efficient without sacrificing accuracy. For example, it is necessary to optimize for reduced memory usage if a LiDAR system needs to detect an object quickly or an image needs to be denoised on a hardware constrained system.

Since a majority of the costs of using a CNN comes from the convolutional layers, which perform a convolution of an input image with a *n-dimensional* learned kernel, a method of increasing the computing and memory efficiency of the convolution operation and kernel respectively is especially important. One method for speeding up CNN computation has been in decreasing the computational complexity of convolution operations through convolution kernel factorization (or decomposition). Factorization can be used to break a higher dimensional convolution into a sequence of effectively lower dimensional convolutions which has a lower computational complexity and approximately the same result. These factorizations can be reflected by a modified neural network architecture as in [1] or by directly factoring already trained convolutional kernels, or by a combination of both [4]. In the former case, a novel architecture is trained while in the latter case, a factorized model is obtained from an already trained model for the specific task.

In this report, both of these methods of kernel factorization are evaluated on the task of denoising images corrupted with additive white Gaussian noise. The evaluation consists of comparing the accuracy on the denoising task and the number of operations and number of parameters stored by the model. We expect the number of operations to be reflective of the speed of the denoising method, and we expect the number of parameters to reflect the amount of memory that the model requires both during storage, and during computation when evaluating the output for an arbitrary input image.

In this report, first we outline the motivation and theory behind both methods of kernel factorization which we evaluate. Then, we describe their implementation details such that others can implement the same models and achieve the same performance. Then, we show empirical results from applying both denoising methods and evaluate the models in terms of parameters and computations in denoising. Finally, we draw a conclusion on the most effective way to reduce the amount of operations and parameters stored by a model while sacrificing the least amount of accuracy. We find that the modified CNN architecture described in [1] performs best in terms of accuracy, number of operations, and parameters required, producing nearly a $7x$ reduction in operations and $10x$ reduction in number of parameters from using a full CNN for denoising. This method shows promise for building memory and computation efficient CNNs.

2. Related Work

Multiple methods of kernel factorization have been proposed with the goal of decreasing the computational com-

plexity of the convolution operation while approximating the convolution with a full kernel. These factorized kernel methods have been evaluated in terms of computations and memory usage on various image classification tasks in [1][3][4] and on various neural network architectures such as ResNet and VGG16 in [2].

In [3], representing a convolutional kernel in a basis of rank-1 kernels is able to achieve a 2.5% speedup with no loss in accuracy. A similar low-rank based factorization method which uses the CP decomposition (which is a generalization of the construction of low-rank approximations of matrices with the SVD to higher dimensional tensors) to decompose $4D$ convolutions into a sequence of four rank r $2D$ convolutions results in an $8.5x$ CPU speedup with only a 1% accuracy drop [4]. By taking advantage of sequential convolution with low-rank tensors, these methods both increase the speed of the convolution operation by reducing computational complexity, and reduce the number of parameters needed to be stored to represent convolution kernels, thereby reducing the amount of memory used by the CNN. Note that both of these factorization methods take kernels from already trained models to factorize in order to produce the resulting factorized CNN, although [4] suggests further training on the factorized model can improve performance.

In [1] (MobileNets), and [2], a modified CNN architecture is proposed based on replacing all convolution operations in the convolutional layers with depthwise separable convolutions, which are convolution operations that can be broken down into a specific type of sequential lower dimensional convolutions. For example, in [1], depthwise separable convolutions turn N different $3D$ convolutions over M channels into M different $2D$ convolutions over each channel independently, and then N independent $1D$ convolutions over these M outputs. These depthwise separable convolutions can then be interpreted as separate, smaller layers in a modified CNN architecture. Note that this method of factorization implies an approximate CNN architecture based on these separable convolutions, rather than factoring an already learned kernel. This new architecture must then be trained for the desired learning task.

Similarly to the directly factorized convolution kernels, the modified CNN architecture requires less parameters and less operations in performing convolution, saving both memory and computing time. In [1], this architecture was evaluated quantitatively in terms of accuracy, Mult-Adds (number of multiplication and addition operations), and parameters, showing improvement in computation and memory efficiency and little decrease in accuracy.

For the specific denoising task which this report applies the CNNs to, multiple algorithms have been proposed which attempt to reconstruct an image which has been corrupted by additive white Gaussian noise (AWGN) of both known and unknown variance. One popular algorithm for denoising is BM3D [8], which breaks an image into blocks and groups similar blocks together, and then takes advantage of the correlation between blocks to estimate the denoised blocks. In this report, this algorithm is used as a baseline for denoising methods and is compared to CNN based denoising.

3. Factorized Convolutional Kernels

In this section the theory behind factorized convolutional kernels is described. We take the example of a single convolution operation over an input image of size $D_F \times D_F \times S$ (i.e. an input $2D$ image with S channels). For our analysis, we establish the convention that the output image is $D_F \times D_F \times T$. This implies that our convolution kernel is of size $D_K \times D_K \times S \times T$, where $D_K \times D_K$ is the size of the convolution in each channel of the input image.

Note that the general convolution operation has computational complexity of $O(D_K \cdot D_K \cdot S \cdot T \cdot D_F \cdot D_F)$. This is because for each pixel in each channel of the input image (of which there are $D_F \times D_F \times S$), $D_K \times D_K$ multiplications and additions must be performed. This must be done T times to produce the T output channels.

3.1. Depthwise separable convolutions

Depthwise separable convolutions, as presented in [1], involve breaking a convolution over a $D_F \times D_F \times S$ input T times into first a "depthwise convolution" followed by a "pointwise convolution". The depthwise convolution individually convolves each of the S channels $D_F \times D_F$ image with a $D_K \times D_K$ convolution kernel, resulting in S channels of size $D_F \times D_F$. The pointwise convolution convolves over the S channels for each point in the image of size $D_F \times D_F$, resulting in a single image of size $D_F \times D_F$. This is done T times in order to generate the T output channels, and an output of size $D_F \times D_F \times T$. This is shown in Figure 1.

More precisely, consider an input U and a desired output V. The formula for a regular convolution is:

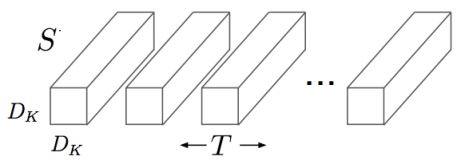
$$V(x,y,t) = \sum_{i,j,s} U(x+i-1, y+j-1, s) \cdot K(i,j,s,t) \tag{1}$$

Where $(i,j) \in \{0...D_K\} \times \{0...D_K\}$, $s \in \{0...S\}$. This is replaced with the following two convolutions, where a depthwise convolution is given by:

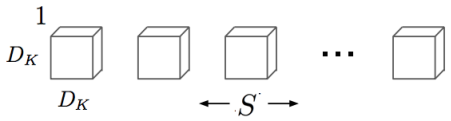
$$\hat{U}(x,y,s) = \sum_{i,j} U(x+i-1, y+j-1, s) \cdot \hat{K}(i,j,s) \tag{2}$$

And the pointwise convolution is given by:

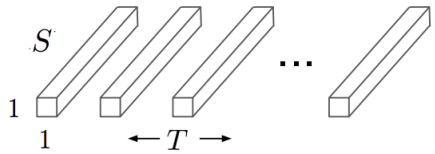
$$V(x,y,t) = \sum_{s} \hat{U}(x,y,s) \cdot \bar{K}(s,t) \tag{3}$$



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 1. Difference between standard convolution and depthwise separable convolution, figure from [1]

To see the advantage of this in terms of memory and computational complexity, note that the factorized convolutional kernels $\bar{K}$ and $\hat{K}$ are of lower dimension. Namely, $\bar{K}$ only requires $S \times T$ parameters to be stored, and $\hat{K}$ only requires $D_K \times D_K \times S$ parameters. With this reduction in parameters also comes a reduction in computational complexity, since the depthwise convolution is $O(D_F \cdot D_F \cdot S \cdot D_K \cdot D_K)$ and the pointwise convolution is $O(D_F \cdot D_F \cdot S \cdot F)$.

3.2. Factorization based on the CP decomposition

Another way of factorizing kernels is described in [4], where the $4D$ convolutional kernel described in equation 1 can be decomposed into four $2D$ convolutional kernels of rank R. The four kernels can be interpreted as first taking the input $D_F \times D_F \times S$ and performing a pointwise convolution to reduce to R channels. Then, each $D_F \times D_F$ image from each of the R channels is convolved horizontally with a $D_K \times 1$ kernel and vertically with a $1 \times D_K$ kernel. Finally, the resulting $D_F \times D_F \times R$ image is pointwise convolved over the R channels to produce a $D_F \times D_F$ output. This is done T times for each output channel, resulting in an output of dimensions $D_F \times D_F \times T$. This is shown in figure 2.

More precisely consider factoring a convolution kernel K as described in equation 1 as

$$K(i,j,s,t) = \sum_{r=1}^{R} K^x(i,r)K^y(j,r)K^s(s,r)K^t(t,r) \quad (4)$$

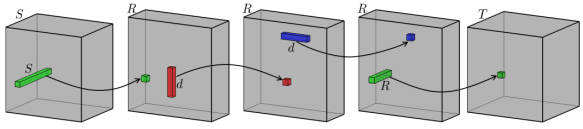


Figure 2. Sequence of rank R convolutions implied by the $2D$ kernels obtained from the CP decomposition, figure from [4]. Note that in this figure, $d = D_K$

Substituting this into equation 1 for kernel $K(i,j,s,t)$, and rearranging the summations, we can see that

$$V(x,y,t) = \sum_{i,j,s} U(x+i-1, y+j-1, s) \cdot ... \sum_{r=1}^{R} K^x(i,r)K^y(j,r)K^s(s,r)K^t(t,r) \quad (5)$$

is equivalent to

$$\sum_{r=1}^{R} K^t(t,r) \cdot \sum_{i} K^x(i,r) \cdot \sum_{j} K^y(j,r) \cdot ... \sum_{s} U(x+i-1, y+j-1, s) \cdot K^s(s,r) \quad (6)$$

Which is just sequential convolutions, if we recursively evaluate the innermost sum of the equation. We can similarly see the advantage of this in terms of memory and computational complexity by examining the dimensions of the factorized kernels obtained from the CP decomposition, K^x, K^y, K^s, K^t. The number of parameters from all of these kernels added together is $S \times R + R \times T + 2 \times D_K \times R$. Similarly, the K^s convolution is $O(D_F \cdot D_F \cdot S \cdot R)$, the K^t convolution is $O(D_F \cdot D_F \cdot T \cdot R)$, the K^y and K^x convolutions are $O(D_F \cdot D_F \cdot D_K \cdot R)$ respectively. When sequentially evaluated, the overall computational complexity is lower than the full convolution.

4. Model Implementation

For a baseline model to modify with the kernel factorizations, we took the U-Net architecture as originally described in [7]. This CNN architecture, originally proposed for image segmentation, generalizes well for many image based learning tasks which also output an image, and was thus determined to be a good CNN architecture in order to denoise images corrupted by AWGN. The specific U-Net used for the denoising task in this report is described in the following section.

4.1. U-Net, dataset and hyperparameters

A number of different U-Net architectures were tested and it was found that a U-Net with a depth of 5 layers (cor-

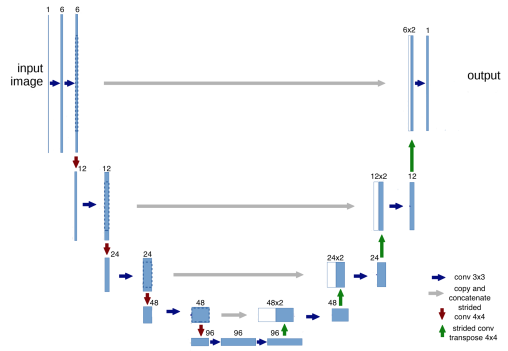


Figure 3. U-Net architecture which was used in the image denoising task. Number on the top of each block describes the number of channels in this data. Each down- or up- sampling convolution reduces the dimensions of the image by a factor of 2 (due to the stride value of 2 in both the convolution and convolution transpose operation). The $\times 2$ channel images result from concatenating previous layers of the U-Net denoised image with further forward layers.

responding to 5 down- and up- sampling convolutions) performed the best. Additionally, the input was convolved to 6 channels, or features, before entering the top level of the U-Net. Each down- or up-sampling convolution was followed by a standard convolution, preserving the dimensions of the image and the number of channels. The architecture is described in figure 3.

In the U-Net architecture described, each down- or up-sampling convolution involved convolving with a 4×4 kernel with a stride of 2, thereby either down- or up-sampling the input to the convolutional layer by a factor of 2. Each down- or up-sampling convolution also increased or decreased (respectively) the number of channels by a factor of 2. The convolutional layers that kept image dimension the same involved convolution with a 3×3 kernel. Each convolution operation was followed by a Batch Normalization (BN) and LeakyReLU (ReLU) operation on the output, which are described at `https://pytorch.org/docs/stable/nn.html`.

The U-Net model was trained on the Berkeley Segmentation Dataset (BSD) [9], which consists of 200 images used for training and 100 images used for testing. All images were converted to grayscale using MATLAB's `rgb2gray` function. Additionally, some of the images were rotated such that they were all $320 \times 480 \times 1$ and could all be trained on in batches. Finally, a noisy version of all grayscale images were obtained by adding Gaussian noise with $\sigma = .1$.

The results reported come from the U-Net model that was trained using batches of 40 full $320 \times 480 \times 1$ noisy images from the training data. This was done instead of training on random crops of the images due to better performance with less training, but it is also possible to train the U-Net on random crops of images and achieve good performance (included in the repository is a model trained on randomized $32 \times 32 \times 1$ croppings of images from the BSD). The model was trained for 180 epochs on a Quadro K6000 GPU with a learning rate of 0.01 to minimize the mean squared error between the original images (without noise) and output of the network from the batch of noisy images. Qualitatively, overfitting does not seem to be an issue and the accuracy on the testing data increases with the number of epochs trained, even beyond 180.

4.2. Depthwise separable convolution model

For the model implemented with depthwise separable convolutions, the U-Net architecture described in figure 3 was modified by taking each convolution operation and replacing it with two convolution operations corresponding to depthwise and pointwise convolution. For the down- and up-sampling convolutions, the depthwise convolution was strided with the same stride as the regular convolution in order to achieve the same amount of down- or up-sampling. As described in [10] and [11], the transposed convolution up-sampling operation can be viewed as a convolution with a fractional stride, supporting the reasoning for replacing these layers with a depthwise transposed convolution and pointwise transposed convolution architecture. The change in architecture is shown in figure 4; the normal convolutional layer architecture (left) is replaced with a sequential depthwise and pointwise convolution (right). The depthwise convolution was implemented with PyTorch's grouped convolution operation allowing us to convolve over each channel separately, while the pointwise convolution was achieved by convolving with a 1×1 kernel spatially over all of the channels.

Unlike in some implementations, the depthwise and pointwise convolutions are not individually followed with BN and ReLU layers, rather the convolutions are sequential and then are followed by BN and ReLU layers.

The modified architecture based on depthwise separable convolutions was then retrained using the same hyperparameters (batch size, epochs, learning rate) as the general U-Net trained for denoising. This is both an advantage and a disadvantage – new architectures can be defined and trained using an architecture based on depthwise separable convolutions and can be tuned to achieve good results through training, however training requires computations and if this is not feasible, this method cannot be used to factor an already trained model into an approximate form that could be used for denoising.

4.3. Factorization of U-Net with CP decomposition

For comparison, the trained U-Net model was factorized using the CP decomposition method. The CP decomposi-

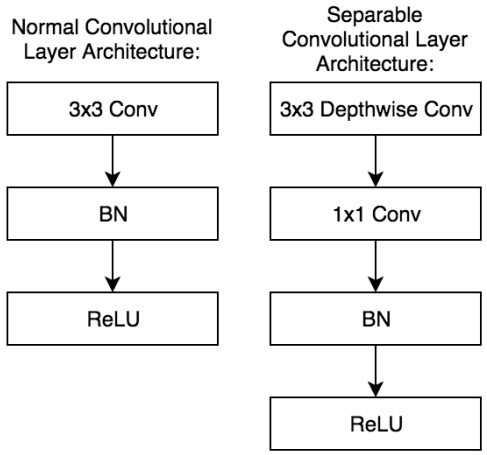


Figure 4. Convolutional layer vs. depthwise separable convolutional layer.

tion of each of the convolutional kernels in each convolution step was computed using the Tensorly package [12], and the corresponding convolution step was replaced with a sequence of four $2D$ convolutions of rank r. Algorithm 1 shows the steps that were taken in order to factorize each of the U-Net convolutional layers. In this algorithm, it is important to notice that the horizontal and vertical convolution are depthwise convolutions (in each channel), so were implemented with grouped convolutions. Similarly, the first and last convolutions are pointwise convolutions which change the number of channels to r. One additional note is that due to the structure of transposed convolutional kernels in Pytorch, we were unable to factorize them effectively with this library, and thus further work must be done in order to fit this method of factorization to a U-Net architecture.

Algorithm 1 CP decomposition of a convolutional layer

Require: Rank: r, $conv2D$ layer to factor: $layer$

1: $pw_{in}, pw_{out}, dw_{vert}, dw_{horiz} = CP(layer.weight, r)$
2: Create $conv2D$ layers with kernel sizes as follows:
$L1 = conv2D(size(pw_{in}))$, $L1.weight = pw_{in}$
$L2 = conv2D(size(dw_{vert}))$, $L2.weight = dw_{vert}$
$L3 = conv2D(size(dw_{horiz}))$, $L3.weight = dw_{horiz}$
$L4 = conv2D(size(pw_{out}))$, $L4.weight = pw_{out}$
3: $layer_{CP} = sequential([L1, L2, L3, L4])$

For strided convolutions, the stride was preserved in the horizontal and vertical convolutions in order to down- or up-sample to the correct dimensions. Additionally, since the amount of channels at the highest level was only 6, factorizing all convolutional kernels to a rank higher than 6 was not possible. This issue can be remedied in future work by altering the rank of the factorization independently for each convolutional layer.

This method can be viewed as a post-processing step on an already trained model. In other words, no training was done after the CP decomposition of the convolution kernels in the U-Net model, although it is suggested in [4] that further tuning through training can result in better performance.

4.4. Profiling

Profiling of the architectures in terms of operations and parameters was done through model examination layer by layer. The number of parameters was calculated by simply accessing the model's trainable parameters in Pytorch and counting them.

To calculate the number of multiplication and addition operations (labeled "operations") the model was inspected and the operations from each layer were calculated based on the size of the input and the layer specifications. In other words, the number of operations was counted by a function which computed the number of adds and multiplications that Pytorch's `conv2d` function would perform, but these adds and multiplications were not directly counted from interaction with the hardware. This was done in order to make the profiling method hardware agnostic. Additionally, the number of operations were calculated only for the convolutional layers, since it the comparison of other layers was not the goal of this report and it is expected that most operations come from the convolutional layers.

Thus, in order to compute the parameters and operations from a model, all that was necessary was access to the model and the size of the input.

5. Results

Accuracy of denoising results are shown for two selected images from the BDS testing dataset in figure 5. The CP decomposed U-Net architecture results in this figure were obtained by decomposing the convolutional kernels from the U-Net into four sequential rank 5 convolution kernels. This number was selected such that the reconstructed image quality was roughly the same as the image quality obtained by other methods.

As shown in the figure, in terms of PSNR and SSIM the U-Net denoising method is comparable to BM3D, the U-Net implemented with the depthwise separable convolution architecture is slightly worse, and the U-Net implemented with the factorized kernels obtained via the CP decomposition is slightly worse than the depthwise separable convolution model. Intuitively this was expected, since the U-Net and depthwise separable convolution U-Net models were directly trained for denoising, while the CP decomposition

Method	#Parameters	#Operations ($\times 10^9$)
Standard U-Net	811100	178.5
DW Separable Convolutions	76968	26.8
CP decomposition of Kernels	351651	127.8

Table 1. Profiling of different CNN architectures

Rank of kernel decomposition	#Parameters	#Operations ($\times 10^9$)
$r = 1$	346299	122.3
$r = 2$	347637	123.7
$r = 3$	348975	125.1
$r = 4$	350313	126.4
$r = 5$	351651	127.8
$r = 6$	352989	129.2

Table 2. Profiling of CP decomposition architectures based on rank of decomposition

model was a mathematical approximation of the full U-Net which had already been trained. Qualitatively however, all of the images denoised via varied U-Net architectures appear practically indistinguishable and much sharper than the result of BM3D. The PSNR and SSIM metrics do not accurately reflect this qualitative difference in image quality.

The number of flops for each of the operations is compared in table 1. The depthwise separable convolution architecture results in nearly a $7x$ reduction in operations and $10x$ reduction in number of parameters from using a full CNN for denoising, while using rank 5 decomposition of the convolutional kernels results in a nearly $1.5x$ reduction in number of operations and $2.3x$ reduction in number of parameters.

An important piece of information is that this accuracy comparison was only carried out for the CP decomposition into rank 5 kernels. The number of parameters, operations, and accuracy of this method depends on our choice for the rank that the kernels from the U-Net are decomposed to. Thus, table 2 and figure 6 show how the CP decomposition method performs as a function of the rank r of the kernel decomposition.

6. Discussion

The results above show that the architecture based on depthwise separable convolutions proposed in [1] performs the best in terms of number of operations, parameters, and accuracy. One drawback to this method is that a model must be trained from scratch based on the modified architecture, but in turn the learned kernels perform better than the direct factorization of already trained kernels with the CP decomposition. However, it is likely that the method based on direct factorization of kernels could perform better with some additional training after the initialization of weights based on the factorization (and could perform well with less training). This could be a direction of future work if the accuracy of the factorized CNN methods needs to be further increased.

A limitation of this work is that the "grouped convolutions" which were used in order to perform all depthwise convolutions (or any spatial convolution individually on each channel) in Pytorch are not optimized to run on a GPU. This has been raised as an issue within the Pytorch community in `https://github.com/pytorch/pytorch/issues/1708` and `https://github.com/pytorch/pytorch/issues/15513`, but currently despite the reduced number of operations and parameters, both factorized kernel methods run significantly slower on a GPU than the standard U-Net. On a CPU, the factorized convolutions are able to produce a speedup, but in this case the training in general is very slow.

One future direction to this work is to apply it to $5D$ convolutions corresponding the $3D$ image inputs. The codebase has already been modified to include a depthwise separable convolution $3D$ U-Net model, and thus with some proper pre-processing of a dataset it should be possible to quickly evaluate convolutions on $3D$ images with the same metrics. Preliminary factorized $3D$ convolution results were not included in this report due to time constraints required to present the results in a interpretable form, but the theory described and results shown for the $2D$ case should generalize to $3D$. In future work, for example, methods using factorized CNNs could be compared to the method in [13], where denoising is done on $3D$ MRI data using a CNN.

Finally, it is likely that these models could be fine-tuned to perform better on the denoising task with better choices for hyperparameters, but this report mainly shows that they are capable of achieving equal performance to algorithms like BM3D designed specifically for denoising. Additionally, this report focuses on the notion that the CNNs implemented with factorized convolutional kernels are able to reduce the computational and memory cost of the CNN without sacrificing much on accuracy. For future uses of these models, it will be necessary to spend more time in fine tuning the model hyperparameters for the specific learning task.

7. Implementation

All code is included at `https://github.com/alexanderbergman7/cnn-factorization`.

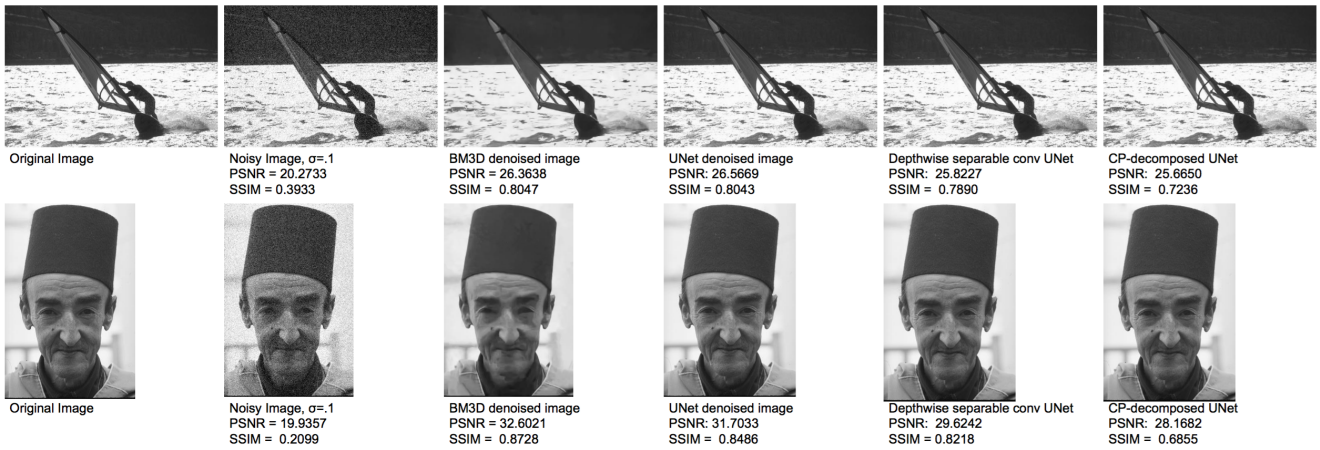


Figure 5. Results

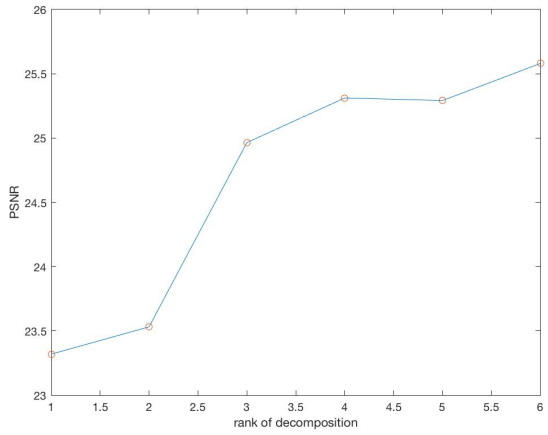


Figure 6. Average PSNR of testing data as a function of rank decomposed to

References

[1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv:1704.04861, 2017.

[2] M. Wang, B. Liu, and H. Foroosh. Factorized convolutional neural networks. arXiv preprint arXiv:1608.04337, 2016.

[3] Jaderberg, M., Vedaldi, A., Zisserman, A. "Speeding up convolutional neural networks with low rank expansions." arXiv:1405.3866, 2014.

[4] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. arXiv preprint arXiv:1412.6553, 2014.

[5] M. Szarvas, U. Sakai, and J. Ogata, "Real-time pedestrian detection using LIDAR and convolutional neural networks" in Proc. 2006 IEEE Intelligent Vehicles Symp., pp. 213-218

[6] Burger, H.C., Schuler, C.J., Harmeling, S., "Image denoising: Can plain neural networks compete with BM3D?" in CVPR 2012, pp. 2392-2399

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation" in Proc. Med. Image Comput. Comput.-Assisted Intervention, 2015, pp. 234241.

[8] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising with block-matching and 3D filtering," Proc. SPIE Electronic Imaging '06, no. 6064A-30, San Jose, California, USA, January 2006.

[9] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A Database of Human Segmented Natural Images and Its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics" Proc. IEEE Intl Conf. Computer Vision, July 2001.

[10] Vincent Dumoulin and Francesco Visin, "A guide to convolution arithmetic for deep learning" arXiv preprint arXiv:1603.07285, 2016.

[11] Wenzhe Shi, Jose Caballero, Lucas Theis, Ferenc Huszar, Andrew Aitken, Christian Ledig, and Zehan

Wang, “Is the deconvolution layer the same as a convolutional layer?” arXiv preprint arXiv:1609.07009, 2016.

[12] Jean Kossaifi, Yannis Panagakis, and Maja Pantic. “Tensorly: Tensor learning in python” arXiv preprint arXiv:1610.09555, 2016.

[13] Liang D, Dou W, Vosters L, Xu X, Sun Y, Tan T, “Denoising of 3D magnetic resonance images with multi-channel residual learning of convolutional neural network” Jpn J Radiol. 2018. https://doi.org/10.1007/s11604-018-0758-8.