Pattern Recognition 89 (2019) 12–21

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Improving CNN linear layers with power mean non-linearity

Chen-Lin Zhang, Jianxin Wu*

National Key Laboratory for Novel Software Technology, Nanjing University, China

ARTICLE INFO

Article history:
Received 28 April 2018
Revised 20 November 2018
Accepted 22 December 2018
Available online 23 December 2018

Keywords:
Non-linearity in deep learning
Pre-trained CNN models
Object recognition
Transfer learning

ABSTRACT

Nowadays, Convolutional Neural Network (CNN) has achieved great success in various computer vision tasks. However, in classic CNN models, convolution and fully connected (FC) layers just perform linear transformations to their inputs. Non-linearity is often added by activation and pooling layers. It is natural to explore and extend convolution and FC layers non-linearly with affordable costs. In this paper, we first investigate the power mean function, which is proved effective and efficient in SVM kernel learning. Then, we investigate the power mean kernel, which is a non-linear kernel having linear computational complexity with the asymmetric kernel approximation function. Motivated by this scalable kernel, we propose Power Mean Transformation, which nonlinearizes both convolution and FC layers. It only needs a small modification on current CNNs, and improves the performance with a negligible increase of model size and running time. Experiments on various tasks show that Power Mean Transformation can improve classification accuracy, bring generalization ability and add different non-linearity to CNN models. Large performance gain on tiny models shows that Power Mean Transformation is especially effective in resource restricted deep learning scenarios like mobile applications. Finally, we add visualization experiments to illustrate why Power Mean Transformation works.

1. Introduction

Convolutional neural network (CNN) is a very successful visual representation learning approach. As a fundamental tool in computer vision, it is essential to improve the CNN's performance and generalization ability. Previous researches mostly focus on developing novel network structures and making the CNN models deeper, e.g., the Inception family [1,2] and the ResNet family [3–6]. In addition to these structures, some researches focus on developing new kinds of non-linear activation functions, such as the ReLU family [7–10].

Beyond these improvements on general CNN structures, some researches also propose new non-linear layers to improve the performance on fine-grained tasks, e.g., Bilinear CNN and some variants [11–17]. These layers are often used after the end of the feature extraction part of CNN models and have immense computational complexity. Thus, they cannot be applied to early layers in a CNN and have difficulty in scaling to large problems. In summary, existing methods have not paid enough attention to the basic building blocks of CNN models: the convolution layers and FC layers.

In this paper, the proposed method is motivated by power mean SVM (abbreviated as PmSVM) [18], which is a non-linear SVM with *linear computational complexity* and has higher accuracy than traditional linear SVM. Based on its non-linear kernel approximation function, we demonstrate a new non-linear transformation: Power Mean Transformation. Power Mean Transformation adds non-linearity to current CNN models, including both input images and output features. The overall structure of our Power Mean Transformation is shown in Fig. 1. Power Mean Transformation can be applied in any place of CNN models. For better efficiency and to achieve the balance between accuracy and speed, Power Mean Transformation is used at two places in this paper: transformation to the input images (which is abbreviated as pm-i) and transformation to output features (which is abbreviated as pm-o). For models with both pm-i and pm-o, we abbreviate it as pm-i&o.

In contrast with those non-linear layers designed for fine-grained tasks, we only add a negligible amount of parameters and almost zero running time to the original model by using Power Mean Transformation. The theoretical complexity of Power Mean Transformation is linear to the number of elements in its input tensor, which is the same as ReLU, and lower than all non-linear layers designed for fine-grained tasks. Compared to those works which develop new activation functions, we add a different kind of non-linearity *before* the linear layer, rather than after the linear layer. With this additional non-linearity, models with Power Mean

* Corresponding author.
E-mail address: wujx2001@gmail.com (J. Wu).

https://doi.org/10.1016/j.patcog.2018.12.029
0031-3203/

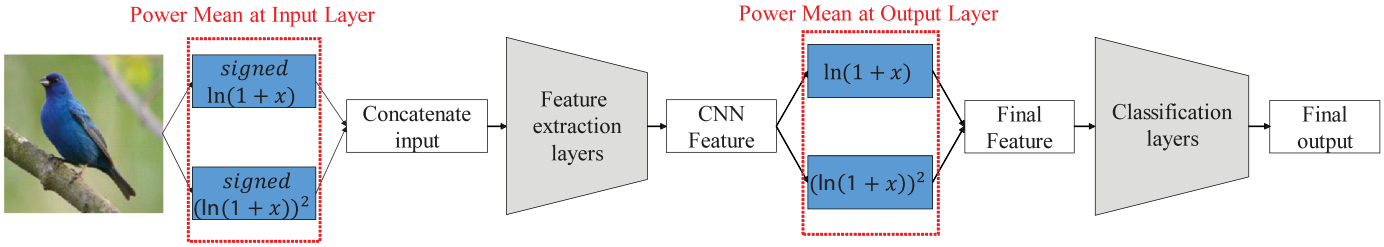

Fig. 1. Overall structure of the proposed Power Mean Transformation. For traditional CNN models, we can add non-linear Power Mean Transformation to both the input images and the output features.

Transformation show better capacity and generalization ability. Visualizations and analyses on Power Mean Transformation illustrate why it works. Furthermore, Power Mean Transformation is easy to be integrated into modern CNN models.

Classification experiments on CIFAR-10 [19] and ImageNet [20] show that Power Mean Transformation improves the performance on various models, especially on not-so-deep models. What's more, transfer learning and object detection experiments verify the good generalization ability of Power Mean Transformation. Finally, we perform experiments by using Power Mean Transformation together with other non-linear layers to show that our transformation adds a different kind of non-linearity to CNN models and can cooperate with other existing non-linear layers.

We summarize our contributions as follows.

- We add *a new pattern of non-linearity* to current CNNs, achieving accuracy improvement with negligible model size and running time increase. With the large performance gain on not-so-deep CNN models, we can directly apply Power Mean Transformation into resource restricted deep learning scenarios.
- We add Power Mean Transformation to the input images. Furthermore, we perform several visualizations to illustrate why our transformation works and to explain our models empirically.

2. Related work

In this section, we will briefly introduce SVM techniques and non-linearity in CNN models.

2.1. Power mean SVM

Support vector machines (SVMs) are learning models which can be used for classification and regression [21]. Researchers have developed various kernel functions for SVM, and achieved significant success on different machine learning and computer vision tasks [22–25].

However, most of these kernels need enormous computational complexity, and cannot be applied to large-scale visual problems such as ImageNet classification. So, some researchers propose improvements on linear kernels. The power mean kernel, which has the same computational complexity as the linear kernel with approximation techniques, is proved to be faster and more effective than the linear kernel [18].

The additive kernel is a family of kernels which can be written as the sum of a scalar kernel function for each dimension, i.e., $\kappa(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} \kappa(x_i, y_i)$. The most popular additive kernels are the histogram intersection kernel (HIK) $\kappa_{HI}(x, y) = \min(x, y)$ and the χ^2 kernel $\kappa_{\chi^2}(x, y) = \frac{2xy}{x+y}$.

In mathematics, a power mean function F_p can be defined as $F_p(x_1, \ldots, x_n) = (\frac{\sum_{i=1}^{n} x_i^p}{n})^{1/p}$ where $p \in \mathbb{R}$ and $x_1, \ldots, x_n \in \mathbb{R}_+$. When $p = -\infty, 0, +\infty$, the power mean function is also well-defined as $\min(x_1, \ldots, x_n)$, $\sqrt[n]{\prod_{i=1}^{n} x_i}$, and $\max(x_1, \ldots, x_n)$, respectively. When $p = -1$ and $-\infty$, the power mean function becomes the χ^2 and HIK kernel, respectively.

With these power mean functions F_p, the power mean kernel family is proposed for two vectors $\mathbf{x}$ and $\mathbf{y}$: $F_p(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} F_p(x_i, y_i)$ [18]. Wu and Yang [18] prove that when $-\infty \le p \le 0$, $F_p(x, y)$ is a positive definite kernel. With the power mean kernel, SVM achieves better performance than linear SVM, and PmSVM (power mean SVM [18]) has linear training and testing complexity.

2.2. Non-linearity in CNN models

Traditional CNNs are mostly composed of these layers: convolution, activation, pooling, normalization and fully connected (FC) layers. For traditional CNNs, non-linearity is only added by activation and pooling layers which follow the linear (convolution and FC) layers. Hence, the common pattern of non-linearity in existing CNNs is one linear operation followed by another non-linear operation. It is natural to explore different patterns of non-linearity in CNN models. In this paper, we explicitly transform the linear operation to non-linear.

Some researches focus on improving the general convolution layer without adding non-linearity to the convolution layer. Dilated convolution is proposed to enlarge the receptive field of CNN models in computer vision tasks like semantic segmentation [26–28]. Some work is proposed to change the convolution kernel shape to better fit the training data and detect small objects [29,30]. However, all these methods are performing linear transformation to the inputs, which lacks non-linear transformation.

Fully connected layers are the original (fundamental) part of neural networks. They are widely used to build up the classification part of traditional CNN models [3,31]. However, there is a serious over fitting problem with fully connected layers. Some specific normalization techniques are proposed to mitigate the problem [32], and some discussions are made on fully connected layers [33,34]. However, fully connected layers are also performing linear transformation to the inputs.

Besides these general CNN layers, some researchers proposed specific non-linear layers for fine-grained classification such as the bilinear CNN [15]. However, the bilinear CNN has great computational complexity. Compact bilinear pooling uses Fast Fourier Transformation (FFT) and count sketch techniques to calculate the outer product in the bilinear CNN efficiently [12]. Kernel pooling approximates RBF kernel and achieves high accuracy on fine-grained datasets [11]. Recently, some researchers have adopted bilinear layers into general CNN models [35]. This type of bilinear layer is restricted to complex network structures, and cannot be applied directly to simple convolution and FC layers.

Some other researches try to integrate known distributions into CNNs [36]. Second-order information is proved to be useful for both fine-grained and general classification tasks with an MPN-COV layer [13]. Li et al. [37] propose faster training methods for MPN-COV layers. However, these methods are only applied af-

ter the CNN feature extraction part, and they also need massive amounts of parameters and high feature dimensionality, e.g., 12,801 for kernel pooling and 32,896 for MPN-COV. The high feature dimension hinders the scalability of CNN models.

Recently, Zoumpourlis et al. [38] propose non-linear convolution filters for CNN-based learning. However, the proposed non-linear filters are based on the Volterra series models, which are infinite order computing models. Although they restricted the order of Volterra-based convolution filters to two, the filters still have huge computational complexity. Thus, non-linear filters are only added at the beginning of current CNN models, and only CIFAR-10 results are provided in [38], which hinders the scalability to larger models and datasets. However, Our Power Mean Transformation only adds negligible parameters, running time and feature dimension (same as the origin CNN models), e.g., 2048 for ResNet-based models, which can be directly applied to large-scale problems.

3. Power mean transformation

In this section, we will introduce Power Mean Transformation. Based on the power mean kernel and related PmSVM techniques, we propose Power Mean Transformation for linear layers by transferring its input non-linearly. For example, the proposed Power Mean Transformation transfers input images non-linearly before feeding them to the first convolution layer, which is *an efficient way to make the linear convolution operation non-linear*.

3.1. Motivation

Modern deep learning classification models can be viewed as having two parts: a feature extraction part and a classification part. The classification part is often composed of FC layers and in some cases activation layers. What classical SVMs perform is just like the classification part of CNN models, i.e., classifying images with extracted features. Since the power mean kernel performs better than the linear kernel with almost zero extra computational and time cost, we are motivated by this phenomenon: can we adopt the power mean kernel into the classification part of CNN models? We will first start from the classification part, i.e., we will first try to add Power Mean Transformation to the output features of CNN models.

3.2. Notations

The following notations are used in the rest of this paper. Given a vector $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, where x_i and y_i are the i-th element of $\mathbf{x}$ and $\mathbf{y}$, respectively. For a traditional linear transformation $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, $\mathbf{W} \in \mathbb{R}^{d\times d}$ is the transformation matrix, and $\mathbf{b} \in \mathbb{R}^d$ is the bias vector (Suppose $\mathbf{x}$ and $\mathbf{y}$ have the same shape, i.e., $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$).

3.3. From SVM kernel to deep learning

After introducing power mean kernel in Section 2.1, given a positive definite kernel κ, the associated feature mapping function ϕ is written as $\phi(\mathbf{x})^T\phi(\mathbf{y}) = \kappa(\mathbf{x}, \mathbf{y})$. To simplify the problem and analyze the problem better, we follow the original problem settings in [18]: There are a set of training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where n is the number of training examples, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$.

For SVM prediction, the formula is:

$$y = \sum_{j=1}^{n} \alpha_j y_j \kappa(\mathbf{x}, \mathbf{x}_j). \tag{1}$$

where $\mathbf{x}$ is the SVM input, κ is the SVM kernel function, $\mathbf{x}_j$ is a support vector, a_j is the weight factor of $\mathbf{x}_j$ and y_j is the label of $\mathbf{x}_j$. According to Wu and Yang [18], the weight vector of the SVM classifier is:

$$\omega = \sum_{i=1}^{n} a_i y_i \phi(\mathbf{x}_i). \tag{2}$$

Concerning a specified training example $\{\mathbf{x}_i, y_i\}$, the gradient G in the coordinate descent method is computed as:

$$G = y_i \omega^T \phi(\mathbf{x_i}) - 1 = y_i \sum_{j=1}^{n} a_j y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) - 1. \tag{3}$$

According to Eqs. (1) and (3), we can see that there is a common part: $\omega^T\phi(\mathbf{x})$. This function plays a vital role in SVM's forward and backward pass. Hence, the authors of [18] propose the computational bottleneck in SVM as a function g:

$$g(\mathbf{x}) = \omega^T \phi(\mathbf{x}). \tag{4}$$

With the computational bottleneck function g, the forward and backward pass of SVM can be rewritten as:

$$\begin{aligned} y &= \sum_{j=1}^{n} \alpha_j y_j \kappa(\mathbf{x}, \mathbf{x}_j) \\ &= g(\mathbf{x}). \end{aligned} \tag{5}$$

$$\begin{aligned} G &= y_i \omega^T \phi(\mathbf{x_i}) - 1 \\ &= y_i g(\mathbf{x_i}) - 1. \end{aligned} \tag{6}$$

For traditional linear kernels, both the forward and backward pass can be computed in $O(d)$ steps. But, when the mapping function ϕ is infinite dimensional or when it cannot be explicitly found, the update step is not feasible or even cannot be computed. Moreover, when we train a SVM, we need to perform backward passes to all training examples. Hence, it is important to approximate the computational bottleneck function with limited time and memory cost when the kernel is non-linear.

Since the input vector of SVM can be normalized to a specific range, e.g., [0,1], by the Weierstrass approximation theorem, we can approximate g on the closed interval (like [0,1]) by a polynomial function to any degree of accuracy, i.e., we can use the explanatory function e to estimate the function accurately: $e(\mathbf{x}) = (1, \mathbf{x}, \ldots \mathbf{x}^{m-1})^T$ for any degree $m \in \mathbb{Z}^+$. It is obvious that e is a polynomial function of $\mathbf{x}$. For efficient power mean kernel approximation, Wu and Yang [18] observe that g is mostly monotone and the curve is similar to a rotated version of quadratic function. Based on the observations, Wu and Yang [18] propose an explanatory variable function e for additive kernels: $e(\mathbf{x}) = (1, \ln(\mathbf{x} + \beta), \ln^2(\mathbf{x} + \beta))$, in which the ln function is applied to every element x_i in $\mathbf{x}$ separately, and β is a positive constant to make the ln function well defined. The explanatory variable function is a second-order function. Although we can use higher-order functions to approximate g with lower error, the second order function is a good balance between the approximation accuracy and the computational complexity. In the PmSVM paper, the authors claim that based on the explanatory variable function, non-symmetric kernel approximation is proposed for calculating the power mean kernel [18]:

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{y}) &= \sum_{i=1}^{d} \kappa(x_i, y_i) \\ &\approx \sum_{i=1}^{d} \begin{pmatrix} 1 \\ \ln(x_i + \beta) \\ \ln^2(x_i + \beta) \end{pmatrix}^T \mathbf{X}^{-1} \begin{pmatrix} \kappa(y_i, c_0) \\ \kappa(y_i, c_1) \\ \kappa(y_i, c_2) \end{pmatrix}. \end{aligned} \tag{7}$$

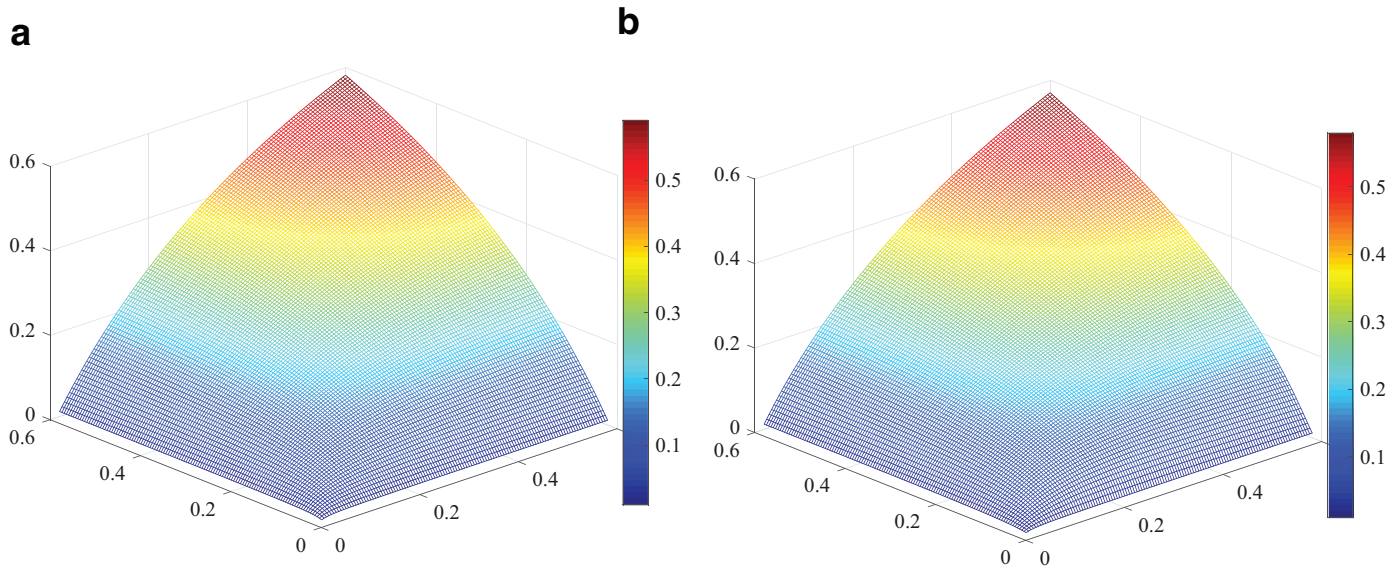

Fig. 2. Meshes showing (a) exact and (b) approximate kernel values. The x and y axes are $x \in (0, 0.6)$ and $y \in (0, 0.6)$, respectively. The z axis shows $\kappa_{\chi^2}(x, y)$ using color codes.

where $\mathbf{X}$ is a constant 3×3 matrix, c_0, c_1, c_2 are constant anchor points in PmSVM. With the constant anchor points defined before, the $\mathbf{X}$ in Eq. (7) is calculated by [18]:

$$\mathbf{X} = \begin{pmatrix} 1 & \ln(c_0+\beta) & \ln^2(c_0+\beta) \\ 1 & \ln(c_1+\beta) & \ln^2(c_1+\beta) \\ 1 & \ln(c_2+\beta) & \ln^2(c_2+\beta) \end{pmatrix}. \tag{8}$$

In [18], the authors originally chose (0.01, 0.06, 0.75) for anchor points since they observed that most feature values are in the range [0.01, 0.1] in the experimented datasets. However, the Chebyshev nodes were later proposed as better anchor points:

$$x_i = \cos\left(\frac{2i-1}{2(m+1)\pi}\right), i = 1, \ldots, m+1, \tag{9}$$

Based on the Chebyshev nodes, if we need n' anchor points, the anchor points can be easily calculated by settings $m = n' - 1$ in Eq. (9). Since $\ln(0)$ is not defined, a small positive β is needed to be added. In PmSVM, the authors set $\beta = 0.05$.

Compared to the traditional SVM kernel calculation process, this formula is a non-asymmetric formula which transfers the non-linear power mean kernel calculation into a linear calculation process.

Fig. 2 shows the results of using Eq. (7) to approximate the χ^2 kernel $\kappa_{\chi^2}(x, y) = \frac{2xy}{x+y}$ when $0 < x < 0.6$, $0 < y < 0.6$. From these two figures, we can easily see that the approximation mesh is almost the same as the mesh which is generated by the original χ^2 kernel, except for some small regions. In fact, the average deviation is 7×10^{-3}, and the average relation deviation is only 3.25%.

Since this approximation is effective and scalable, we want to import it into CNN models. Since Eq. (7) is asymmetric, i.e., if we swap the value $\mathbf{x}$ and $\mathbf{y}$ in Eq. (7), the value of Eq. (7) will be different (although they are almost the same). If we treat the x_i as input variable and treat other variables as constants in Eq. (7), we can see that we only need to transfer x_i to $(1, \ln(x_i+\beta), \ln^2(x_i+\beta))$, the other parts $\mathbf{X}^{-1}(\kappa(y, c_0), \kappa(y, c_1), \kappa(y, c_2))^T$ can be viewed as a constant matrix (z_0, z_1, z_2), and the formula can be rewritten as:

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{y}) &= \sum_{i=1}^{d} \kappa(x_i, y_i) \\ &\approx \sum_{i=1}^{d} \begin{pmatrix} 1 \\ \ln(x_i+\beta) \\ \ln^2(x_i+\beta) \end{pmatrix}^T \begin{pmatrix} z_0 \\ z_1 \\ z_2 \end{pmatrix} \\ &= \sum_{i=1}^{d} z_0 + z_1 \ln(x_i+\beta) + z_2 \ln^2(x_i+\beta), \end{aligned} \tag{10}$$

where x_i, y_i is the ith element of $\mathbf{x}$, $\mathbf{y}$. For SVM prediction, it is performed through Eq. (1). With Eq. (7), it becomes:

$$\begin{aligned} y &= \sum_{j=1}^{n} \alpha_j y_j \kappa(\mathbf{x}, \mathbf{x}_j) + b \\ &\approx \sum_{j=1}^{n} \alpha_j y_j z_0 \sum_{i=1}^{d} 1 + \sum_{j=1}^{n} \alpha_j y_j z_1 \sum_{i=1}^{d} \ln(x_i+\beta) \\ &\quad + \sum_{j=1}^{n} \alpha_j y_j z_2 \sum_{i=1}^{d} \ln^2(x_i+\beta) + b \\ &= \mathbf{c_0}^T \mathbf{1} + \mathbf{c_1}^T \ln(\mathbf{x}+\beta) + \mathbf{c_2}^T \ln^2(\mathbf{x}+\beta) + b, \end{aligned} \tag{11}$$

where y is the predict label, $\mathbf{x}_j$ is a support vector, a_j is the weight factor of $\mathbf{x}_j$, y_j is the label of $\mathbf{x}_j$. In Eq. (11), $\mathbf{c_0} = (a_1 y_1 z_0, \ldots, a_n y_n z_0)$, $\mathbf{c_1} = (a_1 y_1 z_1, \ldots, a_n y_n z_1)$ and $\mathbf{c_2} = (a_1 y_1 z_2, \ldots, a_n y_n z_2)$ summarize those variables that do not depend on $\mathbf{x}$. As shown in Eq. (11), after transforming $\mathbf{x}$ into $(\mathbf{1}, \ln(\mathbf{x}+\beta), \ln^2(\mathbf{x}+\beta))$, the prediction process is a simple linear one. Inspired by this SVM predict process, we can bring this transformation into traditional linear layers in CNN, including the convolution layer and fully connected layer.

For a CNN linear layer, the forward process can be seen as $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$. When we bring non-linear transformation e of $\mathbf{x}$ into the forward process, it becomes:

$$\begin{aligned} \mathbf{y} &= \mathbf{W}(e(\mathbf{x})) + \mathbf{b} \\ &= \mathbf{W}(z_0\mathbf{1}, z_1 \ln(\mathbf{x}+\beta), z_2 \ln^2(\mathbf{x}+\beta)) + \mathbf{b} \\ &= z_0\mathbf{W}\mathbf{1} + z_1\mathbf{W}\ln(\mathbf{x}+\beta) + z_2\mathbf{W}\ln^2(\mathbf{x}+\beta) + \mathbf{b} \\ &= z_1\mathbf{W}\ln(\mathbf{x}+\beta) + z_2\mathbf{W}\ln^2(\mathbf{x}+\beta) + \mathbf{b} + z_0\mathbf{W}\mathbf{1} \end{aligned}$$

$$= z_1\mathbf{W}\ln(\mathbf{x}+\beta) + z_2\mathbf{W}\ln^2(\mathbf{x}+\beta) + (\mathbf{b}+\mathbf{b}_1)$$
$$= \begin{bmatrix} z_1\mathbf{W} \\ z_2\mathbf{W} \end{bmatrix} [\ln(\mathbf{x}+\beta), \ln^2(\mathbf{x}+\beta)] + (\mathbf{b}+\mathbf{b}_1)$$
$$= \mathbf{W}_z[\ln(\mathbf{x}+\beta), \ln^2(\mathbf{x}+\beta)] + (\mathbf{b}+\mathbf{b}_1)\,. \quad (12)$$

In Eq. (12), $\mathbf{b}_1 = z_0\mathbf{W1}$ becomes part of the bias and $\mathbf{W}_z = \begin{bmatrix} z_1\mathbf{W} \\ z_2\mathbf{W} \end{bmatrix}$ is the new transformation matrix for the new non-linear input variable $[\ln(\mathbf{x}+\beta), \ln^2(\mathbf{x}+\beta)]$. Then, with the non-linear transformation of $\mathbf{x}$, the traditional linear layer can be easily modified to be non-linear. Although we can further explore higher order approximation terms such as $\ln^3(\mathbf{x}+\beta)$, our experiments show that they often lead to more parameters and computations, but at the same time lower accuracy in CNN models.

In PmSVM, z_0, z_1, z_2 are constant values and will be calculated before the SVM training process using regression skills, and they are important for approximating the power mean kernel. But according to Eq. (12), z_0, z_1, z_2 can be integrated into the $\mathbf{W}$ matrix of linear layers (as $\mathbf{W}_z$ and $\mathbf{b}_1$), and in the era of deep learning, it is fashionable to learn parameters from a large amount of data directly. So we follow this trend: we let the transformed linear layer learn $\mathbf{W}_z$ directly, i.e., don't manually decide the value of z.

After discussing z, it is easy to perform forward transformation. The constant $\mathbf{b}_1 = z_0\mathbf{W1}$ term can be easily integrated into the bias term in modern CNN linear layers according to Eq. (12), and we do not keep the constant term. For the original PmSVM, features are scaled between [0,1], and PmSVM does not need to handle the situation when features are smaller than zero. For the classification part, extracted features of modern CNN models are larger than or equal to 0 since features are mostly ReLU activated and pooled after convolution layers, and we do not need to handle this case either.

Hence, we use the following transformation: when Power Mean Transformation gets an input $\mathbf{x}$, it transforms $\mathbf{x}$ into $(\ln(\mathbf{x}+\beta), \ln^2(\mathbf{x}+\beta))$, where β is a constant and we will discuss it later in Section 4.1.

With this forward pass, let $\frac{\partial l}{\partial \mathbf{y}}$ be the gradient of Power Mean Transformation's output. Since $y = [\ln(\mathbf{x}+\beta), \ln^2(\mathbf{x}+\beta)]$, by the chain rule we have:

$$\frac{\partial l}{\partial \mathbf{y}} = \frac{\partial l}{\partial[\ln(\mathbf{x}+\beta), \ln^2(\mathbf{x}+\beta)]}\,, \quad (13)$$

$$\frac{\partial l}{\partial \mathbf{x}^T} = \frac{\partial l}{\partial \mathbf{y}^T}\frac{\partial \mathbf{y}}{\partial \mathbf{x}^T}$$
$$= \frac{\partial l}{\partial(\ln(\mathbf{x}+\beta))^T}\frac{\ln(\mathbf{x}+\beta)}{\partial \mathbf{x}^T} + \frac{\partial l}{\partial(\ln^2(\mathbf{x}+\beta))^T}\frac{\ln^2(\mathbf{x}+\beta)}{\partial \mathbf{x}^T}\,. \quad (14)$$

With these forward and backward formulas, the Power Mean Transformation can be easily integrated into CNN models, and use existing optimizers to train it in an end-to-end fashion.

We want to add that Power Mean Transformation can be added into a CNN model very conveniently: only 3 lines of code in Torch.

3.4. Beyond the classification part

Then, we add Power Mean Transformation before the classification part in CNN models. As aforementioned, convolution layers are just the *weight sharing* version of FC layers. We can easily extend Power Mean Transformation to the input of convolution layers in CNN models.

Previously, applying Power Mean Transformation is easy: we do not need to handle negative values. However, for general Power Mean Transformation, we need to handle negative inputs, e.g., normalized image inputs and outputs of convolution layers. In this case, since the convolution layer is also a classical linear layer which can handle negative inputs easily, we use the signed logarithm, i.e., for positive inputs, we just use $\ln(|x|+\beta), \ln^2(|x|+\beta)$, for negative inputs, we use $-\ln(|x|+\beta), -\ln^2(|x|+\beta)$ to replace the original outputs. After this process, we can perform the same non-linear transformation to convolution layers' inputs and then train the model directly by back propagation.

Finally, the algorithm of Power Mean Transformation is in Algorithm 1, in which $\circ$ means element-wise product.

Algorithm 1 Power Mean Transformation.

Forward process:
  Input: Input matrix $\mathbf{x}_{input}$, constant value β
  Output: Output matrix $\mathbf{x}_{output}$
  $sign_{\mathbf{x}} = \text{sign}(\mathbf{x})$
  $e_1(\mathbf{x}) \leftarrow sign_{\mathbf{x}} \circ \ln(|\mathbf{x}|+\beta)$
  $e_2(\mathbf{x}) \leftarrow sign_{\mathbf{x}} \circ (\ln(|\mathbf{x}|+\beta))^2$
  $\mathbf{x}_{output} = \text{concatenate}[e_1, e_2]$
  return $\mathbf{x}_{output}$
Backward process:
  Input: Input gradient $\mathbf{g}_{output}$, input matrix $\mathbf{x}_{input}$, and constant value β
  Output: Output gradient $\mathbf{g}_{input}$
  $sign_{\mathbf{x}} = \text{sign}(\mathbf{x})$
  $\mathbf{g}_{input} = \mathbf{g}_{output} \circ \frac{1}{|\mathbf{x}|+\beta} \circ sign_{\mathbf{x}} + \mathbf{g}_{output} \circ \frac{2\ln(|\mathbf{x}|+\beta)}{|\mathbf{x}|+\beta} \circ sign_{\mathbf{x}}$
  return $\mathbf{g}_{input}$

Because Power Mean Transformation will double the parameter size of next linear layers (since the $\mathbf{W}$ term becomes $\begin{bmatrix} z_1\mathbf{W} \\ z_2\mathbf{W} \end{bmatrix}$), applying Power Mean Transformation at more places in CNN models will directly increase model size and result in unfair comparison. In this paper, we just put Power Mean Transformation at the beginning part of CNN models, i.e., we perform Power Mean Transformation to normalized input images. This choice directly affects the whole model. Furthermore, adding Power Mean Transformation at the beginning part adds few parameters to the current CNN models, since the first convolution layer of current CNN models will have the least output channels among all convolution layers. Finally, we can easily visualize the effect of our Power Mean Transformation.

4. Experiments

In this section, we will show empirical experimental results of the proposed Power Mean Transformation.

4.1. Implementation details

Before diving into implementation details, we first need to decide the value of β. In Power Mean Transformation, to keep extracted features non-negative, and more importantly, when $x = 0$, Power Mean Transformation should lead to the same value as those for small positive and negative values that approach 0, i.e., Power Mean Transformation should be continuous and mostly differentiable for forward and backward pass. Thus, the formula should be $\ln(0+\beta) = -\ln(0+\beta)$. So we choose $\beta = 1$. The β value is same for all Power Mean Transformation, including Power Mean Transformation at input and output.

We will mainly evaluate our transformation on two kinds of popular state-of-the-art models: VGG-based models [31] and ResNet-based models [3]. For all experiments of VGG-based models, we use the Caffe toolbox [39]. For all ResNet-based mod-

els, we use the popular Torch platform for better efficiency. For other transfer learning tasks, we use the original platform of baseline methods. All experiments are conducted on an Ubuntu 14.04 Server with two Intel E5-2680 v4 CPUs and M40 Nvidia GPUs support.

4.2. CIFAR-10 results

We first evaluate our Power Mean Transformation on the CIFAR-10 dataset. For CIFAR-10 experiments, we choose two ResNet models and one wide ResNet model: ResNet-20, pre-activation ResNet-110 (abbr. preResNet-110) and wide ResNet with depth=28 and widen factor=10 (abbr. WRN-28-10). These three models vary in depth and width: ResNet-20 is a relatively shallow model with a comparable error rate on CIFAR-10 while preResNet-110 is a deep model with state-of-the-art error rate. WRN-28-10 widens the ResNet structure and achieves the lowest error rate. We can easily examine the effectiveness of Power Mean Transformation on different models.

Power Mean Transformation will double the parameter of the successive layer in CNN models, e.g., Power Mean Transformation at input and Power Mean Transformation at output will add 0.4 k and 0.64 k parameters to the original ResNet-20 and preResNet-110 models. For fair comparison, we add an extra baseline: For models with Power Mean Transformation at input, we compare it with models which add an extra convolution layer after the first layer. For models with Power Mean Transformation at output, we compare it with models which add an extra convolution layer before the last global averaging pooling layer and fully connected layer. The extra convolution layer will have the same number of output channels as the first or last convolution layer. A batch normalization and ReLU layer will follow all extra convolution layers.

Specifically, for all three models, we add an extra 3×3 convolution layer with 16 input and output channels to compare with Power Mean Transformation at input. The extra convolution layer will add 2.3 k parameters to the models, which exceeds the parameters that Power Mean Transformation at input increases. Successively, we add an extra 3×3 convolution layer with 64 input and output channels to compare with Power Mean Transformation at output. The extra convolution layer will add 36.9 k parameters to the ResNet-20 and preResNet-110 models, which is far beyond the 0.64 k Power Mean Transformation at output parameters. For WRN-28-10 models, since the large dimensionality (640) of WRN-28-10's last convolutional layer's output, it will add about 3M parameters if we add an extra convolution layer. Hence, we do not perform experiments on WRN-28-10's Power Mean Transformation at output.

The hyper-parameter is set same for all three models: batch size 128, 0.0001 weight decay, 0.9 momentum. We will train all models with learning rate 0.1 for 60 epochs, 0.02 for 60 epochs, 0.004 for 40 epochs and 0.0008 for 40 epochs. For wide-resnet models, we set the dropout rate as 0 in all our models. For all models with pm, we report the mean and std of all 5 runs' results. Results are shown in Table 1. Those models denoted by "/" are baseline models and Those models denoted by "*" are models with corresponding extra convolution layers. From those tables, we can easily have these conclusions:

- Power Mean Transformation works both for input images and output features. With pm-i, we can achieve 0.57% and 0.47% performance gain on both ResNet-20 and preResNet-110. The performance boosts are 0.69% and 0.37% on both models with pm-o, respectively. Finally, when we combine pm-i and pm-o, we achieve about 1.2% and 0.8% improvement on both ResNet models.

Table 1
Power Mean Transformation results on the CIFAR-10 dataset on ResNet-20, preResNet-110 and WRN-28-10 models. The best result is marked in bold. The first column shows the mode of Power Mean Transformation. The second column indicates whether it is fine-tuned or trained from scratch. The third, fourth and fifth columns show the Power Mean Transformation results on ResNet-20, preResNet-110 and WRN-28-10, respectively.

pm	train	ResNet-20	preResNet-110	WRN-28-10
/	scratch	8.75%	6.37%	4.00%
pm-i	*(scratch)	8.30 ± 0.21%	6.49 ± 0.07%	4.04 ± 0.07%
	scratch	8.18 ± 0.23%	5.90 ± 0.08%	3.95 ± 0.08%
	*(fine-tune)	8.10 ± 0.19%	6.21 ± 0.10%	4.02 ± 0.10%
	fine-tune	7.98 ± 0.22%	5.80 ± 0.09%	3.90 ± 0.06%
pm-o	*(scratch)	8.08 ± 0.17%	6.43 ± 0.14%	/
	scratch	8.18 ± 0.22%	6.00 ± 0.07%	3.80 ± 0.12%
	*(fine-tune)	7.88 ± 0.15%	6.31 ± 0.09%	/
	fine-tune	7.90 ± 0.21%	5.80 ± 0.08%	3.75 ± 0.09%
pm-i&o	*(scratch)	7.84 ± 0.17%	6.46 ± 0.23%	/
	scratch	7.63 ± 0.13%	5.70 ± 0.10%	3.66 ± 0.08%
	*(fine-tune)	7.78 ± 0.16%	6.22 ± 0.13%	/
	fine-tune	**7.60 ± 0.14%**	**5.55 ± 0.14%**	**3.62 ± 0.09%**

- Fine-tuning also improves the accuracy. For all models we have mentioned before, fine-tuning improves about 0.2% in two ResNet based models.
- With state-of-the-art models, Power Mean Transformation can also improve the accuracy. On top of WRN-28-10 model, pm-i&o can reduce the error rate by 0.38% (about 10% drop in relative error rate).
- Compared to our stronger baseline models, Power Mean Transformation also shows better results. ResNet-20"*" models perform better than ResNet-20 with pm-o models. This phenomenon may due to that ResNet-20 is a relatively small model. But for all other models, especially for preResNet-110 and WRN-28-10, models with Power Mean Transformation have better results, which indicates that the Power Mean Transformation bring new kind of non-linearity to CNN models, not just *deepen* or *widen* current CNN models.

4.3. ImageNet results

We conduct empirical experiments on the ImageNet 2012 dataset. In this section, we also prepare several models:

- Compressed VGG models (ThiNet): Various techniques are developed for compressing CNNs. Among these models, ThiNet achieves state-of-the-art (better than AlexNet) performance with only 1.32M parameters [40]. We choose it as one of the baseline models to evaluate Power Mean Transformation's effectiveness on small networks. We add pm-i, pm-o and pm-i&o to ThiNet to validate the performance empirically.
- Original VGG16 models: The VGG16 model is widely used in many computer vision tasks [31], so we choose VGG16 as a baseline network to evaluate the performance of Power Mean Transformation on models with moderate accuracy. We add pm-i to VGG16.
- ResNet models: The pre-activation version of ResNet achieves state-of-the-art results on ImageNet classification tasks [4]. We choose preResNet-200 (the deepest version of pre-activation ResNet) to evaluate the performance of Power Mean Transformation on deep models. We add pm-i, pm-o and pm-i&o to preResNet-200 to validate the performance empirically.

For ResNet models, first we train the preResNet-200 model with pm-i&o from scratch. As expected, preResNet-200 pm-i&o shows better accuracy than the original preResNet-200 model with a 0.27% margin. That is, training models with Power Mean Transformation from scratch slightly increased its accuracy (compared

Table 2
Power Mean Transformation results on the ImageNet dataset. Note that all models are fine-tuned. The first column shows the mode of Power Mean Transformation. The second column shows results on the ThiNet model, the third column shows results on the VGG16 model and the fourth column shows results on the preResNet-200 model.

pm	ThiNet	VGG16	prePreNet-200
/	59.34%	70.97%	78.34%
pm-i	59.50%	**71.50**%	78.57%
pm-o	60.43%	/	78.58%
pm-i&o	**60.60**%	/	**78.73**%

the model without Power Mean Transformation). Then, based on our observations on CIFAR-10, we choose to perform fine-tuning in the rest experiments. We set the base fine-tuning learning rate to 0.001, momentum to 0.9 and weight decay to 5×10^{-4}. We decay the learning rate by 10 when the training loss does not decrease. Results are shown in Table 2.

Besides the performance comparison, we also compare the number of parameters and running time of different models. Results are in Table 3. From these tables, we can easily have these observations:

- pm-i and pm-o work on both shallow and deep models with the ImageNet dataset. This observation shows Power Mean Transformation is useful for a variety of models. With pm-i, we can achieve 0.16% gain on ThiNet, 0.53% gain on VGG16 and 0.23% gain on preResNet-200. The performance boosts are 1.09% and 0.24% on ThiNet and preResNet-200 with pm-o, respectively. Finally, when pm-i and pm-o are combined, we achieve 1.26% improvement on ThiNet and 0.39% improvement on preResNet-200.
- Power Mean Transformation improves the performance with negligible parameters. For pm-i, the number of added parameters is the same as that of the original first convolution layer, e.g., with VGG-16 models, pm-i only adds 1728 parameters, resulting in a 0.53% performance improvement. For pm-o, the number of added parameters is the same as the original classification layer.
- Power Mean Transformation adds nearly zero running time. For example, with both pm-i and pm-o, preResNet-200 only increases about 4ms running time for the forward pass and 8ms running time for the backward pass, which in total only occupies about 1.5% of the original running time.
- Power Mean Transformation doesn't work for sparsely distributed outputs. In Table 2, The VGG pm-o and VGG pm-i&o models do not perform well. Actually, the pm-o can be only added after the VGG16's fc7 layer. But fc7's output follows a relatively sparse distribution and Power Mean Transformation does not work well on that distribution. However, modern CNN models have used global average pooling to replace fully connected layers [1,3,5], and the distribution of global average pooling layer is relatively dense, which Power Mean Transformation will work, e.g., the results we have shown on ThiNet and preResNet-200.
- Power Mean Transformation works better on relatively shallow models. For ResNet-20 on CIFAR and ThiNet on ImageNet, we get about 1% performance gain, which is significantly higher than the performance boost in other models. This phenomenon may indicate that Power Mean Transformation can be applied in resource restricted deep learning scenarios like mobile applications.

4.4. Transfer learning results

Besides classification performance on the ImageNet dataset, generalization ability is another crucial measurement of CNN models' performance.

We choose scene classification, fine-grained image recognition and object detection tasks to evaluate Power Mean Transformation's generalization ability. For scene classification, we use the MIT Indoor67 dataset [41]. For fine-grained image recognition, we choose the CUB-200 dataset [42]. We choose all models in Section 4.3 as our models in experiments.

For both classification tasks, we set the base learning rate to 0.001, and other hyperparameters remain the same as those in Section 4.3. The learning rate will be divided by 10 when the loss does not decrease. For object detection, we choose the R-FCN method [43] with online hard example mining [44] and pre-trained ResNet-50 as our baseline model, and we use the mean Average Precision (mAP) metric. For detection hyperparameter settings, we follow the traditional hyperparameter settings for all these experiments, i.e., we first reimplement the baseline methods with the original model to get a fair comparison, then we keep all the hyperparameters same, just replacing the original model with our models by adding Power Mean Transformation. We follow the official guide in the R-FCN paper: using online hard example mining (OHEM), a weight decay of 0.0005 and a momentum of 0.9 [44]. We use single-scale training: images are resized such that the scale (shorter side of image) is 600pixels. The results are post-processed by non-maximum suppression (NMS) using a threshold of 0.3 IoU [45]. We also use the python code to perform our experiments. PASCAL VOC 07 and 12 training and validation data are used as training data, and VOC07 test data are used as testing data [46].

The results are shown in Table 4. From these tables, we can easily have these observations as below:

- For both classification tasks, models with Power Mean Transformation perform better than original models, especially on ThiNet based models. The results indicate that all three modes of Power Mean Transformation brings better generalization ability to the model.
- For scene classification tasks, preResNet-200 models do not perform well. It seems that preResNet-200 models are overfitted on MIT Indoor67 dataset.
- For object detection, ResNet-50 pm-o achieves 77.60% mAP while ResNet-50 only achieves 76.60% mAP. Object detection models only use the feature extraction part of CNN classification models, i.e., the classification part is discarded in detection models. These results directly show that pm-o brings better feature extraction ability to the model.

Based on various experiments on Power Mean Transformation, we can conclude that Power Mean Transformation brings better generalization ability than their respective original models.

4.5. Cooperate with other non-linear layers

In this section, we will present joint learning results of Power Mean Transformation plus other non-linear layers.

MPN-COV, which is proposed by Li et al. [13], proves that second-order covariance information is useful for classification tasks, achieving better accuracy on the large-scale ImageNet dataset with different baseline models.

We want to test Power Mean Transformation on top of the MPN-COV method. For simplicity, we choose MIT Indoor67 and ResNet-50 with MPN-COV (MPN-COV-ResNet-50) as our baseline model, and use the same hyper-parameters as in Section 4.4. For

Table 3
Power Mean Transformation model size (number of parameters) and running time comparison on the ImageNet dataset. The second column represents model size and the third to fifth column represent forward time, backward time and total time for each model. Note that for all models, we set the batch size to 32 to get a fair comparison, and we run 20 iterations of each model and take the average running time as final results.

Model	Model size	Forward	Backward	Total
ThiNet	1.32 M	27.00 ms	55.03 ms	82.03 ms
ThiNet w/ pm-i	1.32 M	27.04 ms	55.06 ms	82.07 ms
ThiNet w/ pm-o	1.58 M	27.15 ms	55.10 ms	82.25 ms
ThiNet w/ pm-i&o	1.58 M	27.17 ms	55.11 ms	82.26 ms
VGG-16	138.34 M	188.24 ms	425.95 ms	614.20 ms
VGG-16 w/ pm-i	138.35 M	190.80 ms	426.50 ms	617.30 ms
preResNet-200	64.77 M	352.24 ms	624.00 ms	976.25 ms
preResNet-200 w/ pm-i	64.80 M	355.30 ms	630.26 ms	985.56 ms
preResNet-200 w/ pm-o	66.81 M	354.47 ms	628.95 ms	983.42 ms
preResNet-200 w/ pm-i&o	66.84 M	358.23 ms	632.20 ms	990.43 ms

Table 4
Power Mean Transformation transfer learning results. Note that for all models, different subtable represents different tasks' results. For ThiNet models, the "*" models represent the results from [40].

Scene classification on Indoor67	
Model	Accuracy
ThiNet*	62.69%
ThiNet	64.25%
ThiNet w/ pm-i	64.60%
ThiNet w/ pm-o	65.37%
ThiNet w/ pm-i&o	65.66%
VGG-16	73.80%
VGG-16 w/ pm-i	**74.80**%
preResNet-200	72.62%
preResNet-200 w/ pm-i	73.35%
preResNet-200 w/ pm-o	73.45%
preResNet-200 w/ pm-i&o	73.20%
Fine-Grained classification on CUB-200	
Model	Accuracy
ThiNet*	64.67%
ThiNet	64.90%
ThiNet w/ pm-i	64.93%
ThiNet w/ pm-o	65.90%
ThiNet w/ pm-i&o	66.02%
VGG-16	72.01%
VGG-16 w/ pm-i	72.50%
preResNet-200	79.12%
preResNet-200 w/ pm-i	79.33%
preResNet-200 w/ pm-o	79.64%
preResNet-200 w/ pm-i&o	**79.81**%
R-FCN Detection	
Model	mAP
ResNet-50	77.60%
ResNet-50 w/ pm-o	76.60%

the MPN-COV-ResNet-50 model, we achieve 78.02% accuracy while MPN-COV-ResNet-50 pm-o gets 78.92% accuracy on it.

From these results, we have these findings:

- Power Mean Transformation can effectively perform joint training with other non-linear layers.
- Power Mean Transformation adds a different kind of non-linearity to CNN models, compared to traditional non-linear layers like the bilinear or MPN-COV layer. We can work together with these layers, and achieve even higher accuracy.

5. Visualizations and analyses

This section will give some visualization results on Power Mean Transformation and some theoretical analyses of why Power Mean Transformation works.

For pm-o, it is easy to explain why it works. After the feature extraction part of CNN models, the remaining classification part is often composed with one or a few simple fully connected layers, which can be seen as a simple linear (or non-linear) classifier. As shown by Wu and Yang [18], PmSVM has better accuracy than linear SVM. Hence, when we come into the CNN models, Power Mean Transformation should also perform better than basic fully connected layers with its guided non-linearity, and our experiments validated this hypothesis.

For pm-i, i.e., adding Power Mean Transformation before any convolution layer, it is more complicated to analyze than pm-o because convolution layers have changed the semantic meanings. We use the excitation back propagation (exBP) algorithm [47], which is a powerful visualization algorithm that can model the top-down attention of a CNN classifier for generating task-specific attention maps. The algorithm can tell us the location of CNN's attention in specific layers. In our experiments, since we want to focus on the input images, we set the target layer to the input layer, i.e., we directly watch the focus on input images. For implementation details, we perform the original pre-processing on input images first. Then we do the signed $\ln(1+\mathbf{x})$ and $(\ln(1+\mathbf{x}))^2$ transformation of inputs. After that step, we get three tensors with $H \times W \times 3$ shapes (including the original image tensor).

exBP will return backprop gradients for each location of inputs as results. After getting results of exBP on three image tensors, we use this strategy to visualize our images: for each tensor, we summing it up along the depth dimension. After that, we use the $\frac{2}{3}$ quantile as the threshold of inputs. For those spatial locations whose values are larger than the threshold, we set the pixels as red (i.e., these pixels are highlighted). Otherwise, we set the pixels as their original values. The final figure is in Fig. 3.

Among these figures, we can have the following findings:

- It's obvious to see that the $\ln(1+\mathbf{x})$ and $\ln^2(1+\mathbf{x})$ highlighted maps focus on different areas. The $\ln(1+\mathbf{x})$ part mostly focuses on the continuous area, while the $\ln^2(1+\mathbf{x})$ part mainly focuses on discontinuities. For example, we can look into the bird picture (the third row). We find that $\ln(1+\mathbf{x})$ excitation is almost focused on the bird, while $\ln^2(1+\mathbf{x})$ excitation is mostly on discontinuities, e.g., the borders and tails. Because discontinuity is obviously high frequency information and regions with slowly-changing textures are low-frequency, we conclude that the $\ln(1+\mathbf{x})$ heat map is highlighted for low frequency and the

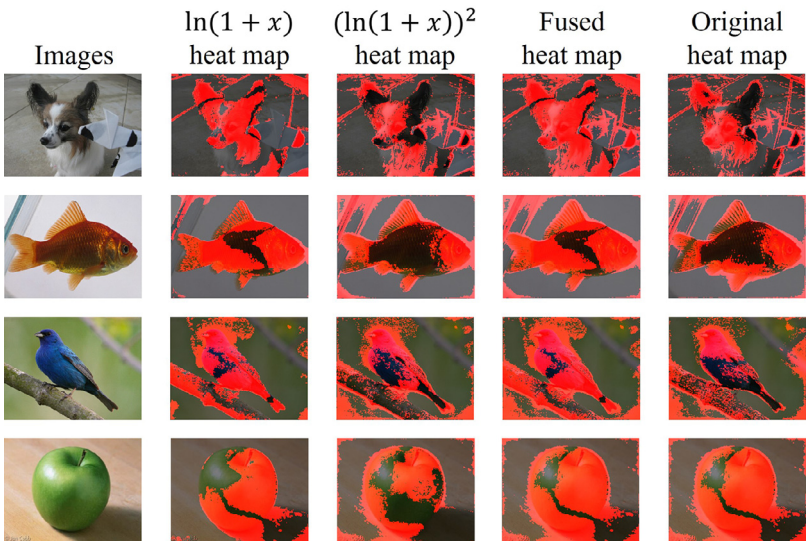

Fig. 3. The exBP heat map of VGG16 baseline model and VGG16 pm-i model. The first column shows the input images. The second and third column show the $\ln(1+\mathbf{x})$ and $(\ln(1+\mathbf{x}))^2$ heat map for VGG16 pm-i model. The fourth column shows the fused heat map of the second and third column. The fifth column shows the heat map of the VGG16 model. For these visualized outputs, the red regions mean that particular model is focused on these regions. The picture is best viewed in the electronic version. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

$\ln^2(1+\mathbf{x})$ heat map is highlighted for high frequency information.

- When we combined the low-frequency and high-frequency parts, the highlighted parts often occupy the whole object. However, for the original VGG16 model, the highlighted area does not include the whole object, as we see in the fourth column and fifth column. This phenomenon may explain why pm-i works.

6. Conclusion

In this paper, we first investigate the Power Mean SVM, which is an effective non-linear SVM with linear computational complexity. Motivated by the efficient asymmetric kernel approximation function in Power Mean SVM, we explore the non-linearity of CNN linear layers, adopting related kernel approximating techniques into CNN, adding *a new pattern of non-linearity* to current CNN models and propose a simple but effective transformation, Power Mean Transformation, to integrate into CNN models. Various results on CIFAR-10 and ImageNet show that our transformation improves the capacity of CNN models with negligible parameter and execution time increase. Furthermore, transfer learning experiments validate that Power Mean Transformation brings generalization ability to CNN models. Finally, Visualization experiments illustrate why Power Mean Transformation works. In the future work, we will further focus on exploring Power Mean Transformation with fewer parameters and applying Power Mean Transformation to more places in CNN models.

Acknowledgement

J. Wu wish to thank the support of National Natural Science Foundation of China (under Grant 61772256 and Grant 61422203).

References

[1] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, Jun. 2015, pp. 1–9.

[2] C. Szegedy, S. Ioffe, V. Vanhoucke, A.A. Alemi, Inception-v4, Inception-ResNet and the impact of residual connections on learning, in: In Proceedings of Association for the Advancement of Artificial Intelligence, San Francisco, CA, Feb. 2017, pp. 4278–4284.

[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, Jun. 2016, pp. 770–778.

[4] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: B. Leibe, J.M.N. Sebe, M. Welling (Eds.), European Conference on Computer Vision,Part IV, LNCS 9908, Springer, Switzerland, Amsterdam, Netherlands, Oct. 2016, pp. 630–645.

[5] S. Xie, R. Girshick, P. Dollar, Z. Tu, K. He, Aggregated residual transformations for deep neural networks, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, Jul. 2017, pp. 5987–5995.

[6] S. Zagoruyko, N. Komodakis, Wide residual networks, in: Proceedings of British Machine Vision Conference, York, UK, Sep. 2016, pp. 87.1–87.12.

[7] C. Djork-Arn, U. Thomas, H. Sepp, Fast and accurate deep network learning by exponential linear units (ELUs), in: Proceedings of International Conference on Learning Representations, San Juan, Puerto Rico, May. 2016, pp. 1–10.

[8] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, FL, Apr. 2011, pp. 315–323.

[9] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on ImageNet classification, in: Proceedings of IEEE International Conference on Computer Vision, Santiago, Chile, Dec. 2015, pp. 1026–1034.

[10] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013, p. 3.

[11] Y. Cui, F. Zhou, J. Wang, X. Liu, Y. Lin, S. Belongie, Kernel pooling for convolutional neural networks, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, Jul. 2017, pp. 2921–2930.

[12] Y. Gao, O. Beijbom, N. Zhang, T. Darrell, Compact bilinear pooling, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, Jun. 2016, pp. 317–326.

[13] P. Li, J. Xie, Q. Wang, W. Zuo, Is second-order information helpful for large-scale visual recognition? in: Proceedings of IEEE International Conference on Computer Vision, Venice, Italy, Oct. 2017, pp. 2070–2078.

[14] T.-Y. Lin, S. Maji, Improved bilinear pooling with CNNs, in: Proceedings of British Machine Vision Conference, London, UK, Sep. 2017, p. In Press.

[15] T.-Y. Lin, A. RoyChowdhury, S. Maji, Bilinear CNN models for fine-grained visual recognition, in: Proceedings of IEEE International Conference on Computer Vision, Santiago, Chile, Dec. 2015, pp. 1449–1457.

[16] G.-S. Xie, X.-Y. Zhang, W. Yang, M. Xu, S. Yan, C.-L. Liu, LG-CNN: from local parts to global discrimination for fine-grained recognition, Pattern Recognit. 71 (2017) 118–131.

[17] X.-S. Wei, C.-W. Xie, J. Wu, C. Shen, Mask-CNN: localizing parts and selecting descriptors for fine-grained bird species categorization, Pattern Recognit. 76 (2018) 704–714.

[18] J. Wu, H. Yang, Linear regression-based efficient SVM learning for large-scale classification, IEEE Trans. Neural Netw. Learn. Syst. 26 (10) (2015) 2357–2369.

[19] A. Krizhevsky, Learning multiple layers of features from tiny images, Technical Report, MSc thesis, University of Toronto, 2009.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (2015) 211–252.
[21] B.E. Boser, I.M. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, in: Proceedings of the annual workshop on Computational Learning Theory, Pittsburgh, PA, 1992, pp. 144–152.
[22] Y. Liu, K. Wen, Q. Gao, X. Gao, F. Nie", SVM based multi-label learning with missing labels for image annotation, Pattern Recognit. 78 (2018) 307–317.
[23] V. Christlein, D. Bernecker, F. Honig, A. Maier, E. Angelopoulou, Writer identification using GMM supervectors and exemplar-SVMs, Pattern Recognit. 63 (2017) 258–267.
[24] S. Paul, M. Magdon-Ismail, P. Drineas, Feature selection for linear SVM with provable guarantees, Pattern Recognit. 60 (2016) 205–214.
[25] T. Malisiewicz, A. Gupta, A.A. Efros, Ensemble of Exemplar-SVMs for object detection and beyond, in: Proceedings of IEEE International Conference on Computer Vision, Barcelona, Spain, Nov. 2011, pp. 89–96.
[26] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, in: Proceedings of International Conference on Learning Representations, San Juan, Puerto Rico, May. 2016, pp. 1–10.
[27] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, Jun. 2015, pp. 3431–3440.
[28] L.-C. Chen, G. Papandreou, F. Schroff, H. Adam, Rethinking atrous convolution for semantic image segmentation, (2017). arXiv: 1706.05587.
[29] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, Y. Wei, Deformable convolutional networks, in: Proceedings of IEEE International Conference on Computer Vision, Venice, Italy, Oct. 2017, pp. 764–773.
[30] Y. Jeon, J. Kim, Active convolution: learning the shape of convolution for image classification, in: IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, Jul. 2017, pp. 4201–4209.
[31] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: Proceedings of International Conference on Learning Representations, San Diego, CA, May. 2015, pp. 1–14.
[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.
[33] M. Lin, Q. Chen, S. Yan, Network in network, in: Proceedings of International Conference on Learning Representations, Banff, Canada, Apr. 2014, pp. 1–10.
[34] C.-L. Zhang, J.-H. Luo, X.-S. Wei, J. Wu, In defense of fully connected layers in visual representation transfer, in: Advances in Multimedia Information Processing, LNCS 10736, Harbin, China, Sep. 2017, pp. 807–817.
[35] Y. Li, N. Wang, J. Liu, X. Hou, Factorized bilinear models for image recognition, in: Proceedings of IEEE International Conference on Computer Vision, Venice, Italy, Jul. 2017, pp. 2079–2087.
[36] Q. Wang, P. Li, L. Zhang, G2DeNet: global gaussian distribution embedding network and its application to visual recognition, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Venice, Italy, Jul. 2017, pp. 2730–2739.
[37] P. Li, J. Xie, Q. Wang, Z. Gao, Towards faster training of global covariance pooling networks by iterative matrix square root normalization, in: The IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, Jun. 2018, pp. 947–955.
[38] G. Zoumpourlis, A. Doumanoglou, N. Vretos, P. Daras, Non-linear convolution filters for CNN-based learning, in: Proceedings of IEEE International Conference on Computer Vision, Venice, Italy, Jul. 2017, pp. 4761–4769.
[39] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R.B. Girshick, S. Guadarrama, T. Darrell, Caffe: convolutional architecture for fast feature embedding, in: Proceedings of the ACM international conference on Multimedia, Orlando, FL, Nov. 2014, pp. 675–678.
[40] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, W. Lin, ThiNet: pruning CNN filters for a thinner net, IEEE Trans. Pattern Anal. Mach. Intell. (2018) in press.
[41] A. Quattoni, A. Torralba, Recognizing indoor scenes, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, Jun. 2009, pp. 413–420.
[42] C. Wah, S. Branson, P. Welinder, P. Perona, S. Belongie, The Caltech-UCSD birds-200-2011 dataset, Technical Report, California Institute of Technology, 2011.
[43] J. Dai, Y. Li, K. He, J. Sun, R-FCN: object detection via region-based fully convolutional networks, in: Advances in Neural Information Processing Systems, Barcelona, Spain, Dec. 2016, pp. 379–387.
[44] A. Shrivastava, A. Gupta, R. Girshick, Training region-based object detectors with online hard example mining, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, Jun. 2016, pp. 770–778.
[45] R. Girshick, J. Donahue, T. Darrell, J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, Jun. 2014, pp. 580–587.
[46] M. Everingham, S.M.A. Eslami, L. Van Gool, C.K.I. Williams, J. Winn, A. Zisserman, The PASCAL visual object classes challenge: a retrospective, Int. J. Comput. Vis. 111 (1) (2015) 98–136.
[47] J. Zhang, Z. Lin, J. Brandt, X. Shen, S. Sclaroff, Top-down neural attention by excitation backprop, in: B. Leibe, J.M.N. Sebe, M. Welling (Eds.), European Conference on Computer Vision,Part IV, LNCS 9908, Springer, Switzerland, Amsterdam, Netherlands, Oct. 2016, pp. 543–559.

Chen-Lin Zhang received his B.S. degree in the department of computer science and technology from Nanjing University, China, in 2016. He is currently working toward the Ph.D. degree in the department of computer science and technology, Nanjing University, China. His research interests are computer vision and machine learning.

Jianxin Wu received his B.S. and M.S. degrees in computer science from Nanjing University, and his Ph.D. degree in computer science from the Georgia Institute of Technology. He is currently a professor in the Department of Computer Science and Technology at Nanjing University, China, and is associated with the National Key Laboratory for Novel Software Technology, China. He has served as an area chair for CVPR, ICCV and AAAI. His research interests are computer vision and machine learning.