Neural Random Subspace

Yun-Hao Cao[a], Jianxin Wu[a,*], Hanchen Wang[b], Joan Lasenby[b]

[a]*National Key Laboratory for Novel Software Technology, Nanjing University, China*
[b]*Department of Engineering, University of Cambridge, UK*

Abstract

Random subspace is the pillar of random forests. We propose Neural Random Subspace (NRS), a novel deep learning based random subspace method. In contrast to previous forest methods, NRS enjoys the benefits of end-to-end, data-driven representation learning, as well as pervasive support from deep learning software and hardware platforms, hence achieving faster inference speed and higher accuracy. Furthermore, as a non-linear component to be encoded into Convolutional Neural Networks (CNNs), NRS learns non-linear feature representations in CNNs more efficiently than previous higher-order pooling methods, producing good results with negligible increase in parameters, floating point operations (FLOPs) and real running time. We achieve superior performance on 35 machine learning datasets when compared to random subspace, random forests and gradient boosting decision trees (GBDTs). Moreover, on both 2D image and 3D point cloud recognition tasks, integration of NRS with CNN architectures achieves consistent improvements with negligible extra cost.

Keywords: random subspace, ensemble learning, deep neural networks

1. Introduction

Deep convolutional neural networks (CNNs) have achieved remarkable advancements in a variety of computer vision tasks [9], such as image classification [20, 35, 10], object detection [8], semantic segmentation [23] and 3D

*Corresponding author

Email addresses: caoyh@lamda.nju.edu.cn (Yun-Hao Cao), wujx2001@nju.edu.cn (Jianxin Wu), hw501@cam.ac.uk (Hanchen Wang), jl221@cam.ac.uk (Joan Lasenby)

recognition [28]. Despite the rapid development of CNNs, forest methods based on random subspaces [11] such as random forests [2] and GBDTs [7] are still the dominant approaches for dealing with vectorized inputs in real-world applications [1]. Benefiting from ensembling predictors with methods such as bagging and boosting, forest methods is capable of making more accurate and robust predictions. However, training these models is computationally expensive, especially for large-scale datasets. Further, forest methods are mostly combinatorial rather than differentiable and they lack the capability of representation learning. On the other hand, CNNs integrate representation learning and classifier learning in an end-to-end fashion, with pervasive software (e.g., deep learning frameworks) and hardware (e.g., GPUs) support, which effectively work on large-scale datasets. To sum up, forest methods benefit from various ensemble mechanisms; while the end-to-end representation learning ability is crucial for deep CNNs. Hence, one question arises: Can we enable such ensemble methods (e.g., random subspaces) with end-to-end representation learning to combine the advantages of ensemble learning and representation learning?

Another interesting aspect is to examine the non-linearity in CNNs. By stacking layers of convolution and non-linearity, CNNs effectively learn discriminative representations. As one standard module in deep CNNs, global average pooling (GAP) summarizes linear statistics of the last convolution layer. Recently, many higher-order pooling (HOP) methods (e.g., [22]) are proposed to learn higher-order, non-linear feature representations to replace GAP and have achieved impressive recognition accuracy. However, these HOP methods suffer from expensive computing costs because of the covariance calculation of very high dimensional matrices. Therefore, another question is: Can we add non-linearity to the linear GAP to achieve both good accuracy and high efficiency?

In this paper, we take a step towards addressing these two questions jointly. We propose a model called Neural Random Subspace (NRS), which is a deep learning based random subspace method. It realizes the random subspace method

[1]https://www. kaggle.com/amberthomas/kaggle-2017-survey-results

in the context ofneural networks, which handles vectorized inputs well (where CNNs do *not* apply) and achieves both higher accuracy (by combining ensemble and representation learning) and faster inference speed (by support from deep learning software and hardware) than conventional random subspace based forest methods, e.g., random subspaces and random forests. We show that such designs are attractive for many real-world tasks dealing with vector inputs.

Furthermore, NRS can be seamlessly installed after the GAP layer at the end of a CNN for image recognition, which non-linearly transforms the output of GAP. As a non-linear component to be encoded into CNNs, NRS is more efficient than HOP methods and achieves higher accuracy than standard GAP with negligible additional cost in terms of model parameters, FLOPs and inference time. Furthermore, NRS can be installed across all layers in a CNN when integrated into Squeeze-and-Excitation (SE) modules [12] and it achieves comparable or better accuracy with fewer model parameters and FLOPs. Aside from 2D image recognition task, we also evaluate NRS on 3D classification tasks where it is used to non-linearly transform the output of the global feature encoders. NRS also brings consistent improvements under various architectures.

Experimental results valid the effectiveness of NRS. We achieve superior performance on 35 machine learning datasets when compared to previous forest methods. On document retrieval datasets, NRS achieves consistent improvements over various baseline algorithms. For 2D image recognition tasks, on the fine-grained benchmarks CUB-200-2011 [40], FGVC-aircraft [25] and Stanford Cars [18], by combining NRS we achieve 5.7%, 6.9% and 7.8% gains for VGG-16, respectively, with negligible increase in parameters, FLOPs and real running time. On ImageNet ILSVRC-12 [33], integration of NRS into ResNet-18 achieves top-1/top-5 errors of 28.32%/9.77%, which outperforms ResNet-18 by 1.92%/1.15% with negligible extra cost. For 3D recognition task on ModelNet40 [42], NRS arises accuracy by 1.1% for PointNet [28] with minor extra complexities.

2. Related Work

2.1. Forest Learning

Forest learning is a powerful learning paradigm which often uses decision trees as its base learners. Bagging and boosting, for instance, are the driving forces of random forests [2] and GBDTs [7], respectively. Random subspaces-based forests [11] select random subsets of features for base learners to construct decision forests. They have become the choices for many industrial applications and data science projects, ranging from face recognition [44] to numerous data science competitions in Kaggle and beyond. Note that *the input to such models are vectors rather than images*, therefore it might not be suitable to use methods such as CNNs to process the data. To accelerate the learning process, ThunderGBM [41] proposes a GPU-based software to improve the efficiency of random forests and GBDTs, especially for large and high dimensional problems. However, they are designed for specific algorithms and hardware, which is lack of generality in comparison with our NRS. With the rapid development of deep learning, there have also been deep forest methods. [45] proposes gcForest, which is a deep forest ensemble with a cascade structure. mGBDTs [6] learn hierarchical distributed representations by stacking several layers of regression GBDTs. However, these methods are not end-to-end trained and thus cannot be accelerated by the deep learning platforms. In contrast, our method integrates random subspace method with end-to-end, data-driven representation learning capabilities with support from existing deep learning software and hardware platforms. Moreover, we train all base learners end-to-end jointly rather than separately as in previous similar method NDF [17]. NDF combines a single deep CNN with a random forest for image classification, where the outputs of the top CNN layer are considered as nodes of the decision tree and prediction loss is computed at each split node of the tree. Our work differs as follows: (i) We implement random subspaces rather than random forests in a novel and easy way, which will be introduced in the next section. (ii) Our method is light-weight and more easily integrated into existing deep learning frameworks.

2.2. Non-linear representations in CNNs

Statistics higher than first-order ones have been successfully used in both classic and deep learning based classification scenarios. The Vectors of Locally Aggregated Descriptors (VLAD) [15] and Fisher Vectors (FV) [27] use non-linear representations based on hand-crafted features. By replacing handcraft features with outputs extracted from CNNs pre-trained on ImageNet [33], these models achieve state-of-the-art results on many recognition tasks [5]. In these designs, representation and classifier training are not jointly optimized and end-to-end training has not been fully studied. [22] proposes a bilinear CNN (B-CNN) that aggregates the outer products of convolutional features from two networks and allows end-to-end training for fine-grained visual classification. [21] proposes an iterative matrix square root normalization (iSQRT) method for fast training of global covariance pooling networks. These works have shown that higher-order, non-linear feature representations based on convolution outcomes achieve impressive improvements over the classic linear representations. However, they suffer from the expensive computational overhead because these methods depend heavily on spectral decomposition or singular value decomposition of very high dimensional covariance matrices. Contrary to previous higher-order methods, our NRS learns non-linear feature representations with only negligible increase in parameters, FLOPs and real running time while achieving higher accuracy.

3. Neural Random Subspace

In this section, we propose the NRS module, which mainly consists of random permutations and group convolutions. We show that it resembles an ensemble of one-level decision trees where each tree learns from a random subset of features (i.e., a random subspace), hence we name it Neural Random Subspace (NRS).

3.1. Network architecture

We first describe the notations in the following and use them consistently in the rest of the paper. We use $\boldsymbol{x} \in \mathbb{R}^d$ to represent a d-dimensional feature

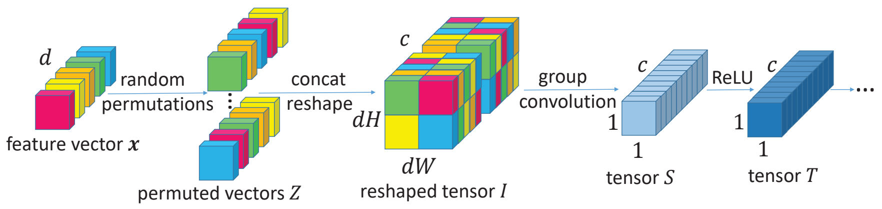


Figure 1: NRS architecture with one group convolution layer.

vector and x_i to represent its i-th element $(i = 1, \ldots, d)$. We denote the depth expansion rate as $nMul$, the expansion height as dH, the expansion width as dW and the number of channels per group in the group convolution as $nPer$.

Our goal is to build a neural classifier based on random subspace, which combines the advantages of both ensemble learning and deep learning. We propose a novel NRS architecture to achieve this goal, as shown in Figure 1. For a d-dimensional feature vector $\boldsymbol{x}$, we first generate m random permutations $\boldsymbol{\sigma}^1, \ldots, \boldsymbol{\sigma}^m$ by reordering the elements in $\boldsymbol{x}$, where $m = dH \times dW \times nMul$. This results in a set of randomly permuted vectors $Z = \{\boldsymbol{z}^t\}$ from $\boldsymbol{x}$ correspondingly, where $\boldsymbol{z}^t$ is generated by t-th permutation $\boldsymbol{\sigma}^t$:

$$\boldsymbol{z}^t = (x_{\boldsymbol{\sigma}^t_1}, \ldots, x_{\boldsymbol{\sigma}^t_d}), \quad t = 1, \ldots, m\,. \tag{1}$$

Then, we concatenate these d-dimensional features into a vector U of $m \times d$ dimensions and reshape it into a 3D $I \in \mathbb{R}^{dH \times dW \times c}$, where $c = nMul \times d$. We denote the entry at the i-th row, j-th column and k-th channel in I as $I(i, j, k)$. it is essentially generated by t-th random permutation $\boldsymbol{\sigma}^t$:

$$t = \lfloor k/d \rfloor \times dH \times dW + (i-1) \times dW + j\,. \tag{2}$$

Hence, $I(i, j, k)$ corresponds to the s-th element in $\boldsymbol{z}^t$:

$$I(i, j, k) = x_{\boldsymbol{\sigma}^t_s}\,, \tag{3}$$

where $s = k \mod d$, and t is calculated by Equation (2). Then, we send the tensor I into a group convolution layer of kernel size (kH, kW), out channel numbers c and group numbers $\lfloor c/nPer \rfloor$ without padding, obtaining a new order-3 tensor S of size (oH, oW, c) followed by ReLU non-linearity. By default we directly use only one group convolution layer with $kH = dH$ and $kW = dW$, thus achieving S of size $(1, 1, c)$, as is done in Figure 1. Further, we can combine multiple group convolution layers with $kH < dH$ and $kW < dW$ to make it deeper, which will be studied in Sec 4.2.1. Then, we add ReLU non-linearity upon S and obtain tensor T of size $(1, 1, c)$. Finally, we feed T into fully connected (FC) layers plus a softmax layer for classification tasks.

3.2. Neural random subspace via CNN implementation

A d-dimensional input vector, can be either a handcraft feature in traditional machine learning or pattern recognition tasks or a learned representation generated by CNNs (e.g., the output of a GAP layer). In our NRS, we first transform it into a tensor I by random permutations.

I includes a set of 2D feature maps $I = \{I^k\}(k = 1, \ldots, c)$. I^k, of size $dH \times dW$, is the k-th feature map of the corresponding channel (the k-th channel). For each feature map I^k, it consists of $dH \times dW$ features, which are randomly selected from the original features. In other words, each feature map I^k corresponds to a random subset of features, that is, a random subspace. Then, each group convolution filter which randomly chooses $kH \times kW \times nPer$ features and the subsequent ReLU layer which acts upon an attribute (i.e., a linear combination of these randomly selected features) can be considered as a one-level oblique decision tree [26]. In Figure 2, we take a group convolution layer with $nPer = 1$ (i.e., a depthwise convolution) as an example for the illustration. We use W^k to denote the k-th depthwise convolution filter, S^k and T^k to denote the k-th channel of S and T $(k = 1, \ldots, c)$ respectively. Then from Equation (3)

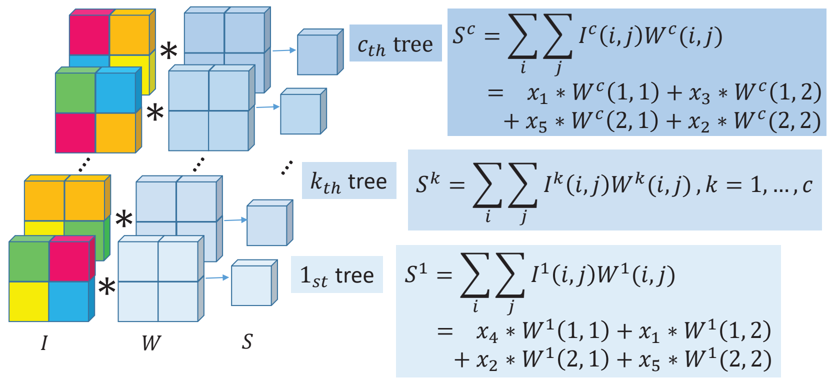

c_{th} tree
$S^c = \sum_i \sum_j I^c(i,j)W^c(i,j)$
$= x_1 * W^c(1,1) + x_3 * W^c(1,2)$
$+ x_5 * W^c(2,1) + x_2 * W^c(2,2)$
k_{th} tree
$S^k = \sum_i \sum_j I^k(i,j)W^k(i,j), k = 1, \ldots, c$
1_{st} tree
$S^1 = \sum_i \sum_j I^1(i,j)W^1(i,j)$
$= x_4 * W^1(1,1) + x_1 * W^1(1,2)$
$+ x_2 * W^1(2,1) + x_5 * W^1(2,2)$
I
W
S

Figure 2: Group convolution makes an ensemble of one-level trees. Each square in different color corresponds to different feature in input feature vector $\boldsymbol{x}$ of 5 dimensions in Figure 1, e.g., the red square corresponds to x_1, etc.

we now have:

$$\begin{aligned} S^k &= \sum_i \sum_j I^k(i,j)W^k(i,j) \\ &= \sum_i \sum_j I(i,j,k) \times W(i,j,k) \\ &= \sum_i \sum_j x_{\boldsymbol{\sigma}_s^t} \times W(i,j,k)\,. \end{aligned} \tag{4}$$

Let $f(\cdot)$ denote the ReLU function, T^k can be computed as:

$$T^k = f(S^k) = \begin{cases} S^k & S^k \geq 0 \\ 0 & S^k < 0 \end{cases} \tag{5}$$

Then, from Equation (5) and Figure 2 we can see that each convolution filter W^k plus the subsequent ReLU resembles a one-level tree which outputs a linear combination of randomly selected features and then a decision based on it. Hence, all convolution filters form an ensemble with c different one-level trees. Actually, if we use 1×1 depthwise group convolution where $kH = kW = 1$, each base learner reduces to a decision stump which learns with a single feature. In conclusion, the random permutation operation acts as resampling and group convolution is used for aggregation in the random subspace. These operations in effect construct random spaces in the context of deep neural networks where each base learner learns from one random subspace. Finally, outputs from those random subspaces are combined for final classification through a combination function and we use FC layers in our NRS.

It is worth noting that decision forests based on random subspaces use bootstrapping to generate feature subsets, in which there is chance that not all features will be utilized [1]. Instead, every feature occurs for the same number of times (exactly m times) in NRS and there is a natural guarantee that every feature will be utilized in the final ensemble. Meanwhile, the injected randomness in NRS guarantees the difference of each random subspace.

Also, when we increase $nMul$, the number of channels c gets larger and we get more group convolution filters. Hence, from Equation (4), more random subspaces are integrated into our ensemble correspondingly. Furthermore, we can increase $nPer$, dH and dW to explicitly increase the number of features utilized in each random subspace, thus increasing the capacity of each base learner. Finally, by stacking more group convolution layers, we can make our network deeper. In all our experiments, we set $dH = dW$ for simplicity, denoted as dH/dW in the rest of this paper. We conduct studies about $nMul, nPer$, dH/dW as well as the number of group convolution layers in Sec 4.2.1 .

4. Experimental Results

In the following section, We will empirically evaluate the effectiveness of our NRS module. On one hand, for vectorized inputs, we compare our method with other competitive forest methods on 34 machine learning classification datasets as well as 1 multivariate regression dataset SARCOS [39] in Sec 4.2. Moreover, we also evaluate our NRS on the challenging document retrieval task Microsoft 10K and Microsoft 30K [30] in Sec 4.3. On the other hand, NRS can be integrated into CNNs for improving non-linear capability either at the end of or across all layers in the network. We conduct experiments on CIFAR-

10 [19], CIFAR-100 [19], fine-grained visual categorization benchmarks and the large-scale ImageNet ILSVRC-12 [33] task with five widely used deep models: MobileNetV2 [34], VGG [35], ResNet [10], Inception-v3 [37] and SENet [12]. Aside from 2D image recognition tasks, we also evaluate our NRS on the 3D recognition task on ModelNet40 [42] in Sec 4.5 with two widely used baselines: PointNet[28] and PoinetNet++[29]. All our experiments were conducted using PyTorch on Tesla M40 GPUs and we will make our code publicly available soon.

4.1. Overview

For machine learning classification and regression datasets, a brief description of them including the train-test split, the number of categories and feature dimensions is given in Table 1. For document retrieval datasets, we use the popular benchmark Microsoft 10K and Microsoft 30K [30] and a brief description is given in Table 5.

For image datasets, CIFAR-10 [19] consists of 50,000 training images and 10,000 test images in 10 classes and CIFAR-100 [19] is just like the CIFAR-10, except it has 100 classes containing 600 images for each class. For fine-grained categorization, we use three popular fine-grained benchmarks, i.e., CUB-200-2011 (Birds) [40], FGVC-aircraft (Aircraft) [25] and Stanford Cars (Cars) [18]. The Birds dataset contains 11,788 images from 200 species, with large intra-class but small inter-class variations. The Aircraft dataset includes 100 aircraft classes and a total of 10,000 images with small background noise but higher inter-class similarity. The Cars dataset consists of 16,185 images from 196 categories. For large-scale image classification, we adopt the ImageNet ILSVRC-12 dataset [33] with 1,000 object categories. The dataset contains 1.28M images for training, 50K images for validation and 100K images for testing. As in [10], we report the results on the validation set.

For the 3D object recognition task, we use ModelNet40 [42] as the benchmark dataset. It contains 40 classes of synthesized CAD models, where the training set has 9,823 objects and the testing set has 2,464 objects. We randomly sample 1,024 points from the mesh faces as the point cloud representation for each object.

4.2. Machine learning datasets

We compare NRS with forest methods, e.g., decision forests based on random subspaces (RSs) [11], random forests (RFs) [2] and GBDTs [7] in terms of accuracy, training/testing time and model size. Furthermore, because we use NRS with 2 FC layers on these machine learning datasets, we also compare it with multi-layer perceptrons (MLP). Our NRS has one more convolution layer than MLP with 2 FC layers (denoted as MLP-2) and hence for fair comparisons we compare with both MLP with 2 FC layers and MLP with 3 FC layers (denoted as MLP-3). Notice that when using dropout [36] at the input layer in MLP (denoted as MLP-D), it can be considered as an ensemble of neural networks trained from different subsets of features and we also compare with it.

Implementation details: We build NRS by 1 group convolution layer and 2 FC layers with batch normalization (BN) in all datasets. We construct MLP-2 and MLP-3 with BN in the same way. For MLP-D, we add dropout at the input layer with $p = 0.8$ as is done in [36] and other settings remain the same as MLP-2 and MLP-3. We split 10% of the training data for validation to determine the total epochs separately for each dataset. We train all networks for 20$\sim$50 epochs, using Adam [16] as optimizer and initializing learning rate to 1e-4. In Table 1, We set $nPer$ to 1 and dH/dW to 3 for all these datasets for simplicity. Considering feature dimensionalities among different datasets, we set different $nMul$ for these datasets to ensure that the product of dimensionalities and $nMul$ is within a relative reasonable interval to save computing resources, as shown in Table 1. For MLP-2, MLP-3, MLP-D, RSs, RFs and GBDTs, we carefully tune the parameters through 5-fold cross-validation on the training set and choose the best parameters for them in each dataset. We report the mean accuracy and standard deviation of 5 trials for all datasets except yeast, which is evaluated by 10-fold cross-validation.

We choose the first 6 datasets satimage, GISETTE, MNIST, letter, USPS and yeast to compare the performance of NRS and several MLP variants, namely

Table 1: Statistics of the machine learning datasets reported in the paper. The above 34 datasets are classfication datasets and SARCOS is a regression dataset.

Datasets	Statistics				NRS setting		
	# Category	# Training	# Testing	# Dim	$nMul$	$nPer$	dH/dW
satimage	6	4435	2000	36	20	1	3
GISETTE	2	6000	1000	5000	10	1	3
MNIST	10	60000	10000	780	16	1	3
letter	26	15000	5000	16	100	1	3
USPS	10	7291	2007	256	30	1	3
yeast	14	1500	917	8	20	1	3
dna	3	1400	1186	180	5	1	3
ijcnn1	2	35000	91701	22	10	1	3
pendigits	10	7494	3498	16	20	1	3
poker	10	25010	1000000	10	50	1	3
protein	3	14895	6621	357	2	1	3
segment	7	1617	693	19	30	1	3
SVHN	10	73257	26032	3072	1	1	3
CIFAR-10	10	50000	10000	3072	1	1	3
connect-4	3	47289	20268	126	5	1	3
SensIT	3	78823	19075	50	20	1	3
splice	2	1000	2175	60	10	1	3
a1a	2	1605	30956	123	10	1	3
a9a	2	32561	16281	123	10	1	3
aloi	1000	75600	32400	128	10	1	3
cod-rna	2	59535	271617	8	50	1	3
covtype	2	406708	174304	54	20	1	3
SUSY	2	3500000	1500000	18	20	1	3
australian	2	483	207	14	20	1	3
breast-cancer	2	478	205	10	50	1	3
fourclass	2	603	259	2	50	1	3
german	2	700	300	24	30	1	3
diabetes	2	537	231	8	50	1	3
heart	2	189	81	13	30	1	3
vehicle	4	592	254	18	30	1	3
sonar	2	145	63	60	10	1	3
glass	6	149	65	9	50	1	3
ionosphere	2	245	106	34	20	1	3
phishing	2	7738	3317	68	10	1	3
SARCOS	-	44484	4449	21	40	1	3

MLP-2, MLP-3 and MLP-D, as shown in Figure 3. For model size, speed and accuracy comparisons, we choose the *highest* dimensional dataset and the 2 *largest* dataset among those 6 and we use some different experimental settings for algorithms in Table 3 . It is hard to make an absolutely fair comparison and for better trade-off between model size, speed and accuracy, we reduce the number of trees for random subspaces (RSs), random forests (RFs) and GBDTs and the parameter $nMul$ for NRS correspondingly. In Table 3, we use the same settings as in Table 2 except that we set $nMul$ to 1, 5 and 50 for NRS for

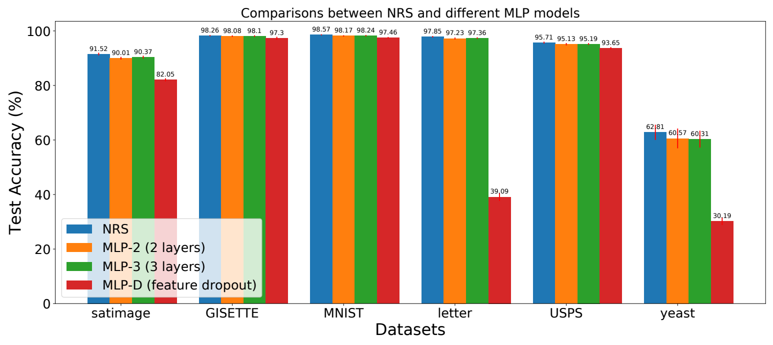


Figure 3: Comparison between NRS and several MLP variants on satimage, GISETTE, MNIST, letter, USPS and yeast. We plot the average accuracy and standard deviation of 5 trails at each bar.

GISETTE, MNIST and letter, respectively, for better model size, speed and accuracy trade-off. Correspondingly, we reduce the number of trees from 500 to 100 for RSs, RFs and GBDTs for faster speed and smaller size. We also use 5-fold cross validation on the train set and choose the best value for other parameters. We record total training time on the train set and inference time on the test set in seconds.

Comparison among different algorithms: Figure 3 shows that MLP-3 achieves comparable performance with MLP-2 and our NRS achieves higher accuracy than both MLP-2 and MLP-3 on all the 6 datasets. Notice that MLP-3 has even more parameters than our NRS and it demonstrates that it is the underlying neural random subspace method rather than the introduced more free parameters to achieve improved performance. In contrast to MLP-D which is inconsistent between training and inference by using a simple approximate average during inference, NRS matains the same structure during both stages and meanwhile achieves consistently better performance. Notice that MLP-D gets very poor results on satimage, letter, yeast and SARCOS which have low feature dimensions (c.f. Table 1) and it indicates that using dropout at the input layer is not suitable for low-dimensional inputs. Also, MLP-2 outperforms MLP-D owing to BN, as pointed out in [13].

Table 2: Accuracy(%) on machine learning benchmarks. We report the average accuracy and standard deviation of 5 trails. NRS and MLP are the results of last epoch. ●/○ indicates that our NRS is significantly better/worse than the corresponding method (pairwise t-tests at 95% significance level). 'N/A' means that no results were obtained after running out 250000 seconds (about 3 days). The last row is the results on the regression dataset SARCOS and * denotes that [38] didn't report the standard deviation.

Datasets	NRS (ours)	MLP	NDF [17]	RSs	RFs	GBDTs
satimage	**91.52±0.31**	90.01±0.31●	89.71±0.31●	90.97±0.08●	91.01±0.35●	89.26±0.04●
GISETTE	**98.26±0.05**	98.08±0.12●	97.24±0.29●	95.72±0.12●	96.98±0.13●	97.18±0.04●
MNIST	**98.57±0.03**	98.17±0.07●	97.29±0.12●	96.83±0.03●	96.96±0.08●	96.56±0.07●
letter	**97.85±0.10**	97.23±0.17●	97.08±0.17●	96.94±0.11●	96.14±0.10●	94.66±0.01●
USPS	**95.71±0.17**	95.13±0.26●	94.99±0.24●	92.81±0.04●	93.80±0.19●	92.83±0.03●
yeast	**62.81±2.61**	60.57±3.45	60.31±3.37●	58.40±2.90	62.81±3.47	60.71±2.35
dna	94.91±0.22	92.56±0.44●	93.12±0.20●	94.60±0.11●	93.64±0.27●	**95.53±0.00**○
ijcnn1	98.34±0.11	**98.55±0.16**○	98.51±0.19	97.15±0.06●	96.76±0.09●	96.17±0.05●
pendigits	**98.03±0.19**	97.11±0.18●	97.55±0.12●	96.79±0.11●	96.46±0.06●	96.13±0.01●
poker	79.28±1.29	70.13±1.60●	68.43±0.41●	74.29±0.30●	64.97±0.26●	**88.13±0.23**○
protein	**69.88±0.31**	67.65±0.19●	69.47±0.17	68.42±0.20●	68.75±0.25●	68.93±0.01●
segment	97.26±0.27	96.45±0.60●	95.04±0.35●	**97.34±0.20**	94.29±0.20●	96.97±0.01●
SVHN	**82.16±0.29**	78.07±3.12●	78.96±0.54●	68.06±0.17●	70.33±0.13●	71.74±0.20●
CIAFR-10	**56.11±0.21**	46.42±2.28●	54.04±0.41●	47.06±0.35●	48.99±0.07●	54.12±0.01●
connect-4	85.70±0.15	84.95±0.20●	**86.32±0.11**○	83.52±0.09●	82.81±0.11●	80.34±0.01●
SensIT	**80.39±0.22**	80.03±0.63	70.30±0.73●	80.13±0.03●	79.89±0.06●	80.12±0.01●
splice	93.25±0.50	88.43±0.81●	91.16±0.22●	**97.03±0.14**○	96.68±0.15○	96.78±0.01○
a1a	**84.33±0.08**	81.86±0.22●	83.18±0.26●	82.13±0.06●	83.06±0.10●	83.61±0.00●
a9a	85.06±0.04	82.54±0.13●	84.84±0.05●	83.52±0.03●	84.77±0.05●	**85.36±0.03**○
aloi	95.76±0.09	95.10±0.08●	N/A	95.61±0.05●	**95.86±0.03**○	N/A
cod-rna	96.71±0.04	96.69±0.05	96.48±0.04●	95.94±0.06●	96.65±0.01●	**96.85±0.01**○
covtype	96.08±0.06	94.88±0.13●	93.62±0.15●	**97.30±0.03**○	95.98±0.02●	95.73±0.01●
SUSY	**80.47±0.01**	80.44±0.01●	79.92±0.01●	79.89±0.02●	80.16±0.01●	80.35±0.00●
australian	87.44±0.75	86.09±0.89●	86.67±1.13	86.09±0.56●	86.86±0.36	**87.92±0.00**
breast-cancer	**97.46±0.57**	96.88±0.79	96.10±0.31●	95.51±0.20●	96.20±0.20●	95.61±0.00●
fourclass	99.54±0.45	99.08±0.67	**99.61±0.10**	97.14±1.70●	95.67±0.15●	98.46±0.00●
german	76.60±0.53	75.20±0.72●	74.00±0.47●	75.33±0.47●	71.40±0.39●	**77.33±0.00**○
diabetes	74.89±0.61	74.11±1.76	75.32±1.02	71.86±0.61●	**75.84±0.32**○	71.86±0.00●
heart	**84.20±1.21**	83.46±2.15	81.98±0.99●	76.79±0.49●	80.99±0.99●	74.07±0.00●
vehicle	**87.48±1.07**	83.54±1.68●	85.14±0.91●	78.58±0.19●	72.52±0.91●	76.38±0.00●
sonar	**92.06±1.42**	88.89±1.42●	86.67±2.15●	79.68±0.63●	74.29±1.19●	85.71±0.00●
glass	**88.62±1.57**	85.23±2.30●	86.46±1.15●	81.85±1.15●	78.46±0.00●	86.15±0.00●
ionosphere	94.53±1.62	94.91±0.96	91.32±1.10●	94.15±0.38	**95.28±0.00**	94.34±0.00
phishing	96.90±0.14	96.77±0.12	96.19±0.13●	**97.05±0.02**○	94.01±0.05●	96.64±0.01●
win/tie/lose		**24/9/1**	**27/5/1**	**28/3/3**	**28/3/3**	**24/3/6**
	NRS (ours)	MLP	ANT [38]	RSs	RFs	GBDTs
SARCOS	**1.23±0.05**	2.36±0.16	1.38*	2.17±0.02	2.37±0.01	1.44±0.01

Table 2 shows that NRS achieves the highest accuracy for the most of times in all classification datasets and the lowest mean square error (MSE) in the regression dataset compared with MLP, RSs, RFs, GBDTs, NDF [17] and ANT [38]. As can be seen, our NRS method significantly outperforms MLP, NDF, RSs, RFs and GBDTs, since the win/tie/lose counts show that our NRS wins for most times and seldom loses and it demonstrate the effectiveness of NRS across datasets with various dimensionalities and sizes. Moreover, it is worth noting that although our method introduces randomness due to random permutations, it achieves a low standard deviation and is very robust, even more stable than MLP.

Table 3: Model size (MB), total inference / training time (s) and accuracy (%) comparison. We report the average results of 5 trails.

	Method	Model size	Time		Accuracy
			Inference	Training	
GISETTE	NRS (ours)	35	0.17	62.51	**97.82**
	RSs	6.6	1.87	57.83	95.60
	RFs	3.6	0.12	**0.67**	96.70
	ThunderGBM RFs	0.6	2.77	24.96	93.60
	GBDTs	**0.2**	**0.01**	181.14	96.70
	ThunderGBM GBDTs	0.6	2.04	18.78	91.79
MNIST	NRS (ours)	9.6	**0.17**	194.45	**98.42**
	RSs	115	5.22	196.12	96.65
	RFs	137	0.31	**2.09**	96.85
	ThunderGBM RFs	6.1	0.76	19.43	93.16
	GBDTs	**1.7**	0.42	2877.78	94.87
	ThunderGBM GBDTs	6.1	0.99	23.66	93.78
letter	NRS (ours)	**3.4**	**0.19**	43.49	**97.78**
	Random Subspaces	151	5.03	4.95	96.50
	Random Forests	106	0.39	**0.38**	96.12
	ThunderGBM RFs	15.9	0.35	16.03	93.29
	GBDTs	4.5	0.27	50.88	92.04
	ThunderGBM GBDTs	15.9	0.33	15.20	92.99

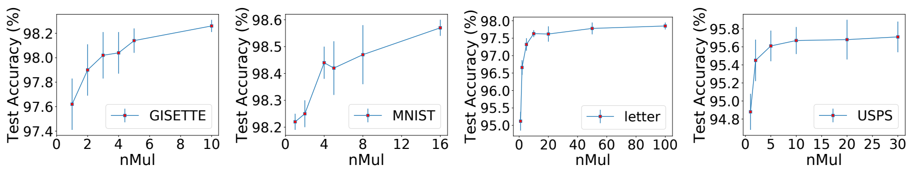
In Tabel 3 we compare the speed and size of NRS with RSs, RFs and GBDTs. Note that although we reduce the number of trees for RFs from 500 to 100 on MNIST, the accuracy drops slightly (from 96.96% to 96.85%) while the model size is reduced by 5 times (from 680M to 137M). Table 3 shows that although GPU-based ThunderGBM can greatly reduce the training time, especially for GBDTs, the inference process seems to have no benefit. Compared to these forest methods, NRS achieves the highest accuracy and the fastest inference speed on MNIST and letter, and also the smallest model size on letter. NRS achieves the highest accuracy on GISETTE but the model size is larger than other forest methods, indicating that NRS may be unfriendly to those datasets with non-sparse high dimensionalities in terms of model size. Note that although we use smaller $nMul$ values in Table 3 than the experiments reported in Table 2, NRS's accuracy in Table 3 are still similar to those in Table 2 (e.g., 97.85 in Table 2 vs. 97.78 in Table 3 on letter). Effect of the sensitivity of NRS's hyperparameters such as $nMul$ will be studied in Sec 4.2.1.

4.2.1. Hyperparameters studies

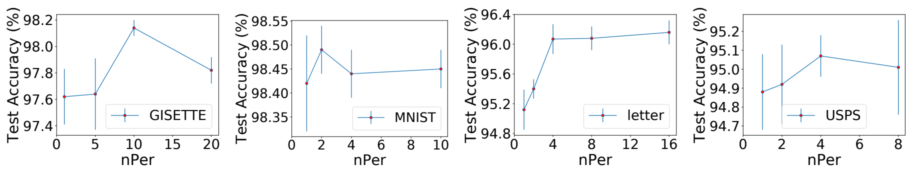
We choose the 4 *largest* datasets among the first 6 machine learning datasets, i.e., GISETTE, MNIST, letter and USPS to study the sensitivity of hyperparameters in our method NRS. Hyperparameters studies include three parts: expansion rate, number of channels per group, expansion height/width and number of group convolution layers.

Expansion rate. As is known in RSs, RFs and GBDTs, we can increase the number of decision trees to boost performance. Similarly, we can increase $nMul$ in NRS to increase the number of trees in our ensemble and we conduct studies about $nMul$. Here we keep other settings the same as before for all experiments. The results in Figure 4a show that when $nMul$ grows, the average accuracy increases and the standard deviation becomes smaller. It indicates that as $nMul$ grows, more trees (random subspaces) are integrated into our model and the performance becomes better and our model gets more robust.

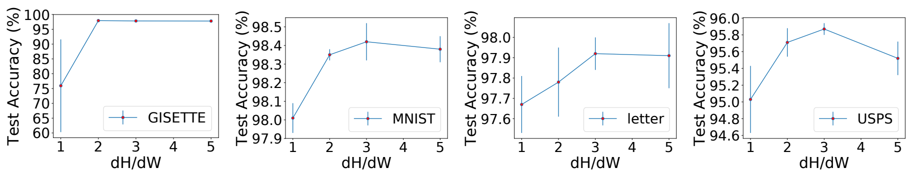
Number of channels per group. We can also increase $nPer$ to increase the number of features utilized in each random subspace. Here we set $nMul$ to 1 for all experiments and other settings remain the same. The results in Figure 4b show that when $nPer$ grows, the test accuracy will increase at first and then it will become stable or slightly decrease. It means that as $nPer$ increases, the capacity of each random subspace and hence the whole model will also increase, thus the accuracy will also increase at first. However, the model is more likely



(a) Classification results using different expansion rates $nMul$.



(b) Classification results using different number of channels per group $nPer$.



(c) Classification results using different expansion size dH/dW.

Figure 4: Hyperparameters studies of $nMul$, $nPer$ and dH/dW on GISETTE, MNIST, letter and USPS (from left to right in each sub figure). We plot the average accuracy and standard deviation of 5 trails at each point.

to overfit with large $nPer$ and model capacity and the performance will not continue to improve. Also, it indicates that we can get better results in Table 2 by choosing appropriate $nPer$.

Expansion height/width. We set $nMul$ to 1, 5, 50 and 20 for GISETTE, MNIST, letter and USPS and we set $nPer$ to 1 for all these datasets. The results in Figure 4c show that when dH/dW is very small, i.e., equals 1, the result is bad, especially for GISETTE. When dH/dW grows, the result becomes better and will not continue to improve when it grows beyond 3. Therefore, 2 or 3 is a good choice for dH/dW in terms of accuracy and efficiency and we use $dH/dW = 3$ in all our experiments in this paper for simplicity.

Number of group convolution layers. We also study the effect of more group convolution layers and we compare the results of NRS with 1 group convolution layer and 2 group convolution layers. Experimental results show that both settings achieve comparable results on these 4 datasets while more group convolution layers brings more computing overhead. Hence, we use NRS with 1 group convolution layer in all our experiments in this paper.

4.2.2. Ablation Study

Notice that the random permutation operation is a linear operation and can be equivalently implemented by a FC layer with sparse weight matrix and we conduct ablation studies on this special FC layer. This FC layer has weight matrix $W^{FC} \in \mathbb{R}^{d \times md}$, which maps input $\boldsymbol{x} \in \mathbb{R}^d$ to $\boldsymbol{y} \in \mathbb{R}^{md}$, where $m = dH \times dW \times nMul$ defined as before. Then we have $y_j = x_i$, where $i = \boldsymbol{\sigma}_{j \bmod m}^{\lfloor j/m \rfloor}$. Correspondingly, the weight matrix is highly sparse with $W_{ij}^{FC} = 1$ $(j = 1, \cdots, md)$ and other weights are set to 0, i.e., each output unit is connected to one particular input unit with weight 1. Hence, our random permutation operation can be considered as a FC layer with two special properties: (i) sparse initialization: the initial weight is highly sparse and specified by the generated permutation. (ii) freezed weight: the weight matrix of this layer is fixed during training. To further validate the effectiveness of our random permutation operation in NRS, we implement the random permutation by a FC layer and ablate its two special properties.

We choose 1 large dataset MNIST, 1 medium dataset satimage and 2 small datasets german and heart. Table 4 shows that the row (d) achieves superior performance over other strategies consistently on all the 4 datasets and it shows that sparse initialization and freezed weight properties in the random permutation has its effectiveness. Also, it is worth noting that the row (c) achieves higher accuracy than both (a) and (b) on the 2 small datasets german and heart, which indicates that the over-parameterization in the FC layer makes the model more likely to overfit and the weight freezing strategy is beneficial, especially on small datasets. Moreover, contrary to the implementation by a FC layer, the random permutation operation has no parameters and small FLOPs and hence it is much more efficient than a FC layer. In conclusion, the random permutation operation in NRS achieves both higher accuracy and efficiency than a FC layer implementation.

Table 4: Classification results evaluated on satimage, GISETTE, MNIST and letter. Starting from our baseline, we gradually add sparse initialization and freezed weight scheme in the FC layer (equivalently implementation of random permutation) in our NRS for ablation studies.

	Scheme		Accuracy			
	sparse init.	freezed weight	MNIST	satimage	german	heart
(a)	×	×	98.21±0.10	91.01±0.42	72.20±0.78	80.99±0.60
(b)	✓	×	98.18±0.08	91.06±0.16	71.87±1.05	81.73±1.44
(c)	×	✓	98.03±0.05	91.08±0.49	74.60±1.06	82.96±0.92
(d)	✓	✓	**98.57±0.03**	**91.52±0.31**	**76.60±0.53**	**84.20±1.21**

4.3. Document Retrieval Datasets

Aside from the machine learning datasets used before, we apply NRS into the challenging document retrieval task, which also has vectorized inputs.

Table 5: The characteristics of document retrieval datasets used in our experiments: number of queries, documents, relevance levels, features and year of release.

	Queries	Doc.	Rel.	Feat.	Year
Microsoft 10K	10000	1200k	$\{0, 1, 2, 3, 4\}$	136	2010
Microsoft 30K	31531	3771k	$\{0, 1, 2, 3, 4\}$	136	2010

Implementation details: In our experiments, we used two widely used benchmark datasets, Microsoft 10K and Microsoft 30K [30]. Each query-document pair is represented with a feature vector. The groundtruth is a multiple-level relevance judgment, which takes five values from 0 (irrelevant) to 4 (perfectly relevant). The basic statistics of each dataset are listed in Table 5, which includes the number of queries, the number of documents, the relevance level of ground truth, the number of features and the year of release. The metrics we adopted is nDCG [14] and we report the results with different cutoff values 1, 3, 5, 10 and 20 and 50 to show the performance of each method at different positions.

In our experiments, 4 typical listwise ranking methods ListNet [4], Approx-NDCG [31], RankCosine [32], WassRank [43] as well as 1 pariwise ranking method RankNet [3] are used as our baselines. Following prior work [4, 43], a simple 1-layer feed-forward neural network (a dropout rate of 0.01) with the Sigmoid activation function is used as the ranking function. For NRS, we set $nMul$ to 2, $nPer$ to 1 and dH/dW to 3 in all our experiments in this section. Following [43], we used the L2 regularization with a decaying rate of 1e-3 and the Adam [16] optimizer with a learning rate of 1e-3. In particular, each dataset has been randomly partitioned into five equal sized subsets. In each fold, three subsets are used as the training data, the remaining two subsets are used as the validation data and the testing data, respectively. We use the training data to learn the ranking model, use the validation data to select the hyper parameters based on nDCG@10, and use the testing data for evaluation. Finally, we report the ranking performance based on the averaged evaluation scores across five folds with 100 epochs.

Comparison among different algorithms: As can be seen from Table 6 and Table 7. NRS achieves consistent improvements under various baseline ranking models on both Microsoft 10K and Microsoft 30K. The experimental results demonstrate the superior performance of NRS compared with the baseline methods on document retrieval tasks.

Table 6: Performance of different models on Microsoft 10K. The best result of each setting is indicated in bold. ▲ denotes our method.

Method	Microsoft 10K					
	NDCG@1	NDCG@3	NDCG@5	NDCG@10	NDCG@20	NDCG@50
RankNet [3]	0.3061	0.3270	0.3414	0.3699	0.4006	0.4498
RankNet+NRS ▲	**0.3704**	**0.3701**	**0.3783**	**0.4019**	**0.4306**	**0.4745**
ListNet [4]	0.3482	0.3514	0.3596	0.3805	0.4093	0.4553
ListNet+NRS ▲	**0.3881**	**0.3849**	**0.3924**	**0.4140**	**0.4421**	**0.4850**
ApxNDCG [31]	0.1328	0.1507	0.1662	0.1961	0.2369	0.3171
ApxNDCG+NRS ▲	**0.3988**	**0.3906**	**0.3972**	**0.4180**	**0.4449**	**0.4852**
RankCosine [32]	0.3798	0.3874	**0.3971**	0.4174	0.4435	0.4839
RankCosine+NRS ▲	**0.3990**	**0.3891**	0.3959	**0.4177**	**0.4464**	**0.4882**
WassRank [43]	0.2552	0.2610	0.2715	0.1947	0.3268	0.3877
WassRank+NRS ▲	**0.3917**	**0.3861**	**0.3926**	**0.4138**	**0.4401**	**0.4815**

Table 7: Performance of different models on Microsoft 30K. The best result of each setting is indicated in bold. ▲ denotes our method.

Method	Microsoft 30K					
	NDCG@1	NDCG@3	NDCG@5	NDCG@10	NDCG@20	NDCG@50
RankNet [3]	0.2997	0.3211	0.3392	0.3692	0.4008	0.4506
RankNet+NRS ▲	**0.3628**	**0.3652**	**0.3767**	**0.4002**	**0.4281**	**0.4733**
ListNet [4]	0.3927	0.3904	0.3975	0.4190	0.4439	0.4833
ListNet+NRS ▲	**0.4029**	**0.3950**	**0.4023**	**0.4235**	**0.4499**	**0.4912**
ApxNDCG [31]	0.1178	0.1405	0.1585	0.1918	0.2340	0.3152
ApxNDCG+NRS ▲	**0.4067**	**0.3969**	**0.4025**	**0.4235**	**0.4495**	**0.4904**
RankCosine [32]	0.3864	0.3913	0.4012	0.4220	0.4466	0.4861
RankCosine+NRS ▲	**0.4118**	**0.4012**	**0.4084**	**0.4287**	**0.4546**	**0.4947**
WassRank [43]	0.4041	0.3924	0.3983	0.4181	0.4442	0.4861
WassRank+NRS ▲	**0.4244**	**0.4122**	**0.4160**	**0.4333**	**0.4557**	**0.4922**

4.4. Computer Vision Datasets

We then move from vectorized inputs to image data and we evaluate NRS in CNN architectures for both 2D image recognition tasks in this section and 3D recognition tasks in the next section. For image recognition tasks, NRS is used after GAP to non-linearly transform the GAP output vector at the end of the network and we evaluate it on both fine-grained visual categorization tasks in Sec 4.4.1 and the large-scale ImageNet ILSVRC-12 in Sec 4.4.2. Moreover, we demonstrate that NRS can further be installed across all layers in CNNs (e.g., SENet [12])and we evaluate it on CIFAR-10, CIFAR-100 and ImageNet ILSVRC-12 in Sec 4.4.3.

4.4.1. Fine-grained Visual Categorization

We then evaluate NRS in CNN architectures for image recognition. NRS is used after GAP to non-linearly transform the GAP output vector at the end of the network. First, this section evaluates NRS with ResNet-50 [10] and VGG-16 [35] on the Birds, Aircraft and Cars datasets. We compare our method with baseline models and one representative higher-order pooling method.

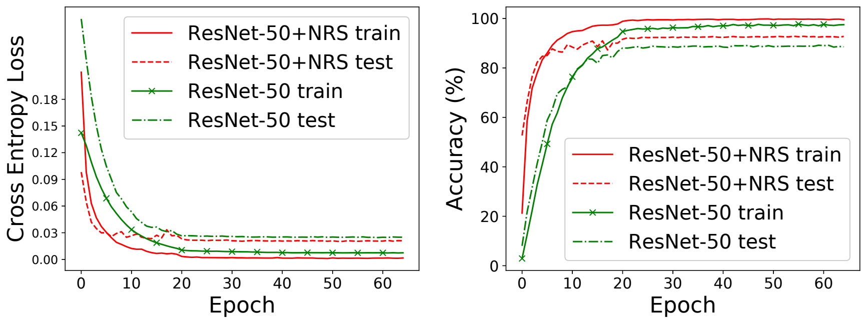
Implementation details: For fair comparisons, we follow [22] for experimental setting and evaluation protocal. We crop 448×448 patches as input

Table 8: Comparison of representation dimensions, parameters, FLOPs, inference time per image (ms) and accuracy (%) on fine-grained benchmarks. The inference time is recorded with batch size of 1 on CPU and GPU. ▲ denotes our method.

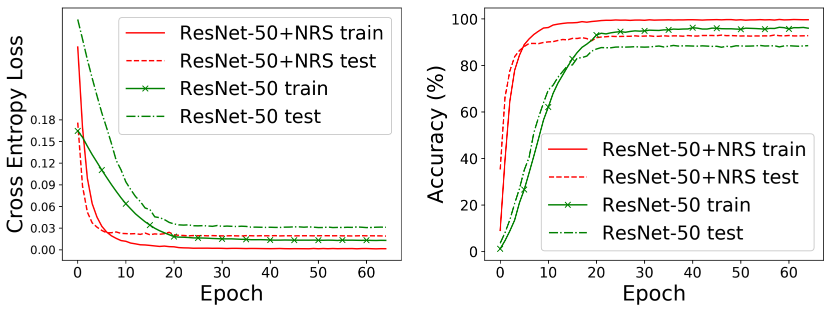
Method	#Dim	#Params	#FLOPs	Inference Time		Accuracy		
				CPU	GPU	Birds	Aircraft	Cars
ResNet-50	2K	23.92M	16.53G	540.48	28.16	84.0	88.6	89.2
ResNet-50+NRS ▲	4K	26.70M	16.53G	541.57	28.38	**86.7**	**92.8**	**93.4**
VGG-16	0.5K	15.34M	61.44G	644.18	28.24	78.7	82.7	83.7
B-CNN [22]	262K	67.14M	61.75G	856.46	31.90	84.0	84.1	90.6
VGG-16+NRS ▲	3K	17.11M	61.44G	645.19	28.58	**84.4**	**89.6**	**91.5**

images for all datasets. For baseline models, we replace the 1000-way softmax layer of ResNet-50 pretrained on ImageNet ILSVRC-12 [33] with a k-way softmax layer for finetuing, where k is the number of classes in the fine-grained dataset. We replace all FC layers of pretrained VGG-16 with a GAP layer plus a k-way softmax layer to fit 448×448 input. We fine tune all the networks using SGD with batch size of 32, a momentum of 0.9 and a weight decay of 0.0001. We train the networks for 65 epochs, initializing the learning rate to 0.002 which is devided by 10 every 20 epochs. For NRS models, we replace the 1000-way softmax layer of pretrained ResNet-50 with our NRS module, specifically, random permutations, 1 group convolution layer and a k-way softmax layer (k is number of classes), which is called ResNet-50+NRS. Here we set $nMul$, $nPer$ and dH/dW to 2, 64 and 3, respectively. Moreover, we also use the pretrained VGG-16 as our backbone network to construct VGG-16+NRS in a similar way, except that we set $nMul$ to 6 considering different feature dimensionalities. We fine tune our models under the same setting as the baseline models. These models integrating NRS are trained end-to-end as the baseline models.

Comparison among different algorithms: Table 8 shows that our NRS method achieves significant improvements compared to baseline models, with negligible increase in parameters, FLOPs and real running time. It is worth mentioning that VGG-16+NRS achieves $7.2\%, 8.3\%$ and 9.3% relative improvement over baseline models on Birds, Aircraft and Cars, respectively. Besides, our NRS performs consistently better than B-CNN [22] on all the 3 datasets under the VGG-16 architecture despite using much fewer parameters, FLOPs and real running time. Furthermore, the learning curves in Figure 5 shows that NRS can greatly accelerate the convergence and achieves better results both in accuracy and convergence speed than baseline methods (the red curves vs. the green curves). It indicates that NRS can effectively learn non-linear feature representations and achieves good results on fine-grained recognition.



(a) Learning curves of Aircrafts



(b) Learning curves of Cars

Figure 5: Loss and accuracy learning curves. Both ResNet-50 and ResNet-50+NFL are trained under the same setting.

4.4.2. ImageNet ILSVRC-12

We then evaluate NRS on the large-scale ImageNet ILSVRC-12 task and also NRS is used after GAP at the end of the network.

Table 9: Error rate (%, 1-crop prediction) comparison on ImageNet ILSVRC-12 under different architectures. ▲ denotes our method.

Method	#Params	#FLOPs	Top-1 / Top-5 error
Original ResNet-50[1]	25.56M	4.14G	23.85 / 7.13
ResNet-50+NRS ▲	29.98M	4.14G	**23.12** / **6.62**
Original ResNet-18[1]	11.69M	1.82G	30.24 / 10.92
ResNet-18+NRS ▲	13.82M	1.83G	**28.32** / **9.77**
Original MobileNetV2[1]	3.50M	0.33G	28.12 / 9.71
MobileNetV2+NRS ▲	3.88M	0.33G	**27.42** / **9.39**

[1] `https://pytorch.org/docs/master/torchvision/models.html`

Implementation details: We train a ResNet-50+NRS model from scratch on ImageNet, which is described in Sec 4.4.1 except that the last layer is a 1000-way softmax layer. The images are resized with shorter side=256, then a 224×224 crop is randomly sampled from the resized image with horizontal flip and mean-std normalization. Then, the preprocessed images are fed into ResNet-50+NRS model. We train ResNet-50+NRS using SGD with batch size of 256, a momentum of 0.9 and a weight decay of 1e-4 for 100 epochs. The initial learning rate starts from 0.1, and is devided by 10 every 30 epochs. A ResNet-18+NRS model is constructed and trained in a similar way, except that we set $nMul$ and $nPer$ to 4 and 32, respectively. For MobileNetV2 [34], we set $nMul$ and $nPer$ to 1 and 32, respectively. We train the network using SGD with batch size of 256, a momentum of 0.9 and a weight decay of 4e-5 for 150 epochs. We initialize the learning rate to 0.05 and use cosine learning rate decay.

Comparison with baseline methods: Table 9 shows that NRS produces 0.70%, 1.92% and 0.73% top-1 error (1-crop) less than the original MobileNetV2, ResNet-18 and ResNet-50 model, respectively, with negligible increase in parameters and FLOPs. It indicates that our NRS method is also effective for large-scale recognition, achieving better performance consistently under various architectures.

4.4.3. NRS across all layers

Motivated by the Squeeze-and-Excitation (SE) method [12], we use NRS to replace all the SE modules. We conduct experiments on CIFAR-10 [19], CIFAR-100 [19] and the ImageNet ILSVRC-12 task. We compare our method with baseline methods and SENet under various architectures.

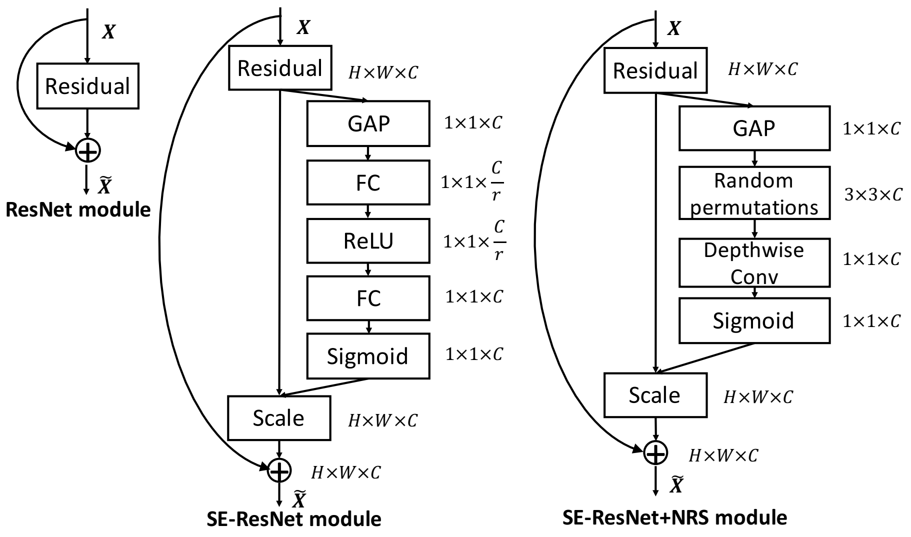


Figure 6: The schema of the original Residual module (left), the SE-ResNet module (middle) and the SE-ResNet+NRS module (right).

Implementation details: We replace the 2 FC layers in each SE block with NRS, specifically, random permutations and 1 group convolution layer followed by sigmoid activation, which is called SENet+NRS, as shown in Figure 6. In all our experiments in this section, we set $nMul$ and $nPer$ to 1 and dH/dW to 3 for SENet+NRS and the reduction ratio is set to 16 for SENet as is done in [12]. For CIFAR-10 and CIFAR-100, we use ResNet-20 [10], ResNet-50 [10] and Inception-v3 [37] as the backbone network. Mean subtraction, horizontal random flip and 32×32 random crops after padding 4 pixels on each side were performed as data preprocessing and augmentation. We train all networks from scratch using SGD with 0.9 momentum, a weight decay of 5e-4 and batch size of 128 for 350 epochs. The initial learning rate starts from 0.1 using cosine learning

Table 10: Comparison of params, FLOPs and accuracy (%) on CIFAR-10 and CIFAR-100 under various architectures. ▲ denotes our method.

Method	CIFAR-10			CIFAR-100		
	#Params	#FLOPs	Acc.	#Params	#FLOPs	Acc.
original ResNet-20	0.27M	41.62M	92.75	0.28M	41.63M	69.33
SE-ResNet-20	0.27M	41.71M	93.28	0.28M	41.72M	70.35
SE-ResNet-20+NRS ▲	0.28M	41.71M	**93.73**	0.28M	41.72M	**70.38**
original ResNet-50	23.52M	1311.59M	95.78	23.71M	1311.96M	80.41
SE-ResNet-50	26.04M	1318.42M	95.59	26.22M	1318.79M	**81.57**
SE-ResNet-50+NRS ▲	23.67M	1313.56M	**96.05**	23.86M	1313.93M	81.48
original Inception-v3	22.13M	3411.04M	94.83	22.32M	3411.41M	79.62
SE-Inception-v3	23.79M	3416.04M	95.60	23.97M	3416.41M	80.44
SE-Inception-v3+NRS ▲	22.23M	3412.85M	**95.67**	22.42M	3413.22M	**80.54**

Table 11: Comparison of parameters, FLOPs, inference time per image (ms) and error rate (%, 1-crop prediction) comparison on ImageNet ILSVRC-12 under SENet architectures. The inference time is recorded with batch size of 1 on both CPU and GPU. ▲ denotes our method.

Method	#Params	#FLOPs	Inference Time		Top-1 / Top-5 error	
			CPU	GPU	reported in [12]	our results
Original ResNet-50[1]	25.56M	4.14G	465.39	21.07	24.80 / 7.48	23.85 / 7.13
SE-ResNet-50	28.07M	4.15G	581.32	35.82	23.29 / 6.62	**22.68** / **6.30**
SE-ResNet-50+NRS ▲	25.71M	4.14G	523.97	32.80	- / -	22.89 / 6.57

[1] `https://pytorch.org/docs/master/torchvision/models.html`

rate decay. For ImageNet, we follow the same setting as in [12]. The images are resized with shorter side=256, then a 224×224 crop is randomly sampled from the resized imgae with horizontal flip and mean-std normalization. We use SGD with a momentum of 0.9, a weight decay of 1e-4, and batch size of 256 and the initial learning rate is set to 0.15 and decreased by a factor of 10 every 30 epochs. Models are trained for 100 epochs from scratch.

Comparison with baseline methods: Table 10 shows that under ResNet-20, SENet+NRS achieves the highest accuracy on CIFAR-10 and CIFAR-100 with negligible increase in parameters and FLOPs. For ResNet-50 and Inception-v3 backbone, SENet+NRS achieves comparable or better accuracy than original

SENet despite using fewer parameters and FLOPs, further confirming the effectiveness of NRS.

Table 11 shows that under ResNet-50, SENet+NRS achieves fewer parameters, FLOPs and real running time than original SENet while maintaining comparable accuracy. SENet+NRS also achieves higher accuracy than the baseline method with negligible increase in parameters and FLOPs. It indicates that NRS can be integrated not only at the end of a CNN as shown in the previous sections but also across all layers in a CNN to learn non-linear mapping effectively.

4.5. 3D Recognition

We then move from 2D image recognition to 3D recognition in this section.

Common types of 3D objects/scenes include point clouds, polygonal meshes, volumetric grids and multiple/depth images (Figure. 7). A point cloud is a set of points in space sampled from object surfaces, usually collected by 3D sensors such as LiDAR. Other representation types include polygon meshes, volumetric grids and multiple view images. Among these 3D representations, we

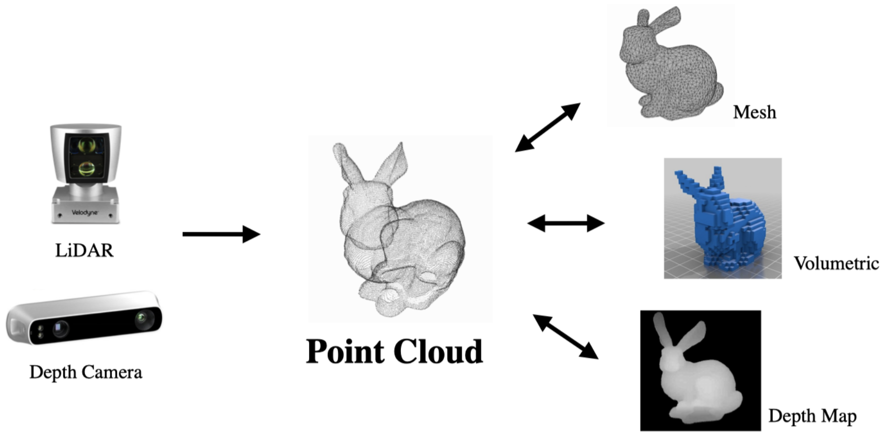


Figure 7: Various Representations of 3D Object

are particularly interested to utilize point clouds for 3D understanding tasks, because of two reasons. First, a point cloud is the closest representation to raw collected sensor data. It encodes full information from sensors, without any quantization loss (in volumetric representations) or projection loss (in multi-view representations). Second, a point cloud is quite neat in form, just a collection of points, which avoids the combinatorial irregularities and complexities of meshes (e.g. choices of polygons, polygon sizes and connectivities), and thus is easier to learn from. The point cloud is also free from the necessity to choose resolution as in volumetric representations, or projection viewpoint in multi-view images.

For point cloud based 3D object recognition tasks, NRS is used after the global feature encoders and we evaluate its effectiveness on ModelNet40 [42] under PointNet [28] and PointNet++ [29]. We compare baselines enhanced with NRS modules and baseline methods with the vanilla versions.

Implementation details: For both baselines, we train them using Adam[16] optimizer with L2 regularisation[24]. The initial learning rate is set to be 1e-3, and it is decayed by a factor of 0.7 for every 20 epochs. We train the model for 200 epochs. We use a batch size of 24 and the momentum in batch normalisation is 0.9. We represent each object with 1024 points, for PointNet++ we also utilise the surface normal of each point. For the NRS modules, we set $nMul$, $nPer$, and dH/dW to 1, 32 and 3, respectively and the training protocols remain the same as baseline models. During training process, we randomly drop, rescale and translate the point cloud for the augmentation.

Table 12: Comparison of parameters, FLOPs, inference time per point cloud objects (ms) and accuracy (%) comparison on ModelNet40 under PointNet and PointNet++ architectures. The inference time is recorded with batch size of 1 on both CPU and GPU. ▲ denotes our method.

Method	#Params	#FLOPs	Inference Time		Accuracy
			CPU	GPU	
PointNet [28]	3.47M	0.45G	27.06	7.18	89.2
PointNet+NRS ▲	3.77M	0.45G	28.69	7.25	90.3
PointNet++ [29]	1.48M	0.87G	234.16	248.03	91.9
PointNet+++NRS ▲	1.78M	0.87G	235.98	248.97	92.3

Comparison with baseline methods: Table 12 shows that both PointNet+NRS and PointNet+++NRS achieve higher accuracy than the corresponding baseline methods with negligible increase in parameters, FLOPs and inference time. We prove that NRS can be integrated within the CNN models that working not only on 2D images, but also 3D objects efficiently and effectively.

5. Conclusions

We proposed a deep learning based random subspace method NRS. We introduced the random subspace method into deep learning with random permutations acting as resampling and group convolutions acting as aggregation, where each base learner learns from a random subset of features. NRS can handle vectorized inputs well and can be installed into CNNs seamlessly both at the end of the network and across all layers in the network for both 2D and 3D recognition tasks. On one hand, it enriches random subspaces with the capability of end-to-end representation learning as well as pervasive deep learning software and hardware support. On the other hand, it effectively learns non-linear feature representations in CNNs with negligible increase in parameters, FLOPs and real running time. We have successfully confirmed the effectiveness of NRS on standard machine learning datasets, popular CIFAR datasets, challenging fine-grained benchmarks, the large-scale ImageNet dataset as well as 3D recognition dataset ModelNet40. In the future, we will continue exploration on combining deep learning and traditional ensemble learning algorithms to better understand the relation between different approaches. Furthermore, we will extend NRS to handle datasets with high dimensional sparse features, as well as small datasets.

References

[1] Breiman, L.: Bagging predictors. Machine learning **24**(2), 123–140 (2001)

[2] Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)

[3] Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: The International Conference on Machine Learning. pp. 89–96 (2005)

[4] Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to rank: From pairwise approach to listwise approach. In: The International Conference on Machine Learning. pp. 129–136 (2007)

[5] Cimpoi, M., Maji, S., Vedaldi, A.: Deep filter banks for texture recognition and description. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 3828–3836 (2015)

[6] Feng, J., Yu, Y., Zhou, Z.H.: Multi-layered gradient boosting decision trees. In: Advances in neural information processing systems. pp. 3551–3561 (2018)

[7] Friedman, J.H.: Greedy function approximation: a gradient boosting machine. The Annals of Statistics **29**(5), 1189–1232 (2001)

[8] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 580–587 (2014)

[9] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., Chen, T.: Recent advances in convolutional neural networks. Pattern Recognition **77**, 354–377 (2018)

[10] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 770–778 (2016)

[11] Ho, T.K.: The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(8), 832–844 (1998)

[12] Hu, J., Shen, L., Albanie, S., Sun, G., Wu, E.: Squeeze-and-excitation networks. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 7132–7141 (2018)

[13] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: The International Conference on Learning Representations. pp. 1–11 (2015)

[14] Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Transactions on Information Systems **20**(4), 422–446 (2002)

[15] Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 3304–3311 (2010)

[16] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: The International Conference on Learning Representations. pp. 1–9 (2015)

[17] Kontschieder, P., Fiterau, M., Criminisi, A., Buló, S.R.: Deep neural decision forests. In: The IEEE International Conference on Computer Vision. pp. 1467–1475 (2015)

[18] Krause, J., Stark, M., Deng, J., Fei-Fei, L.: 3D object representations for fine-grained categorization. In: ICCV Workshop on 3D Representation and Recognition. pp. 554–561 (2013)

[19] Krizhevsky, A., Hinton, G.E.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)

[20] Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)

[21] Li, P., Xie, J., Wang, Q., Gao, Z.: Towards faster training of global covariance pooling networks by iterative matrix square root normalization. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 947–955 (2018)

[22] Lin, T.Y., RoyChowdhury, A., Maji, S.: Bilinear CNN models for fine-grained visual recognition. In: The IEEE International Conference on Computer Vision. pp. 1449–1457 (2015)

[23] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 3431–3440 (2015)

[24] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: The International Conference on Learning Representations. pp. 1–9 (2019)

[25] Maji, S., Rahtu, E., Kannala, J., Blaschko, M., Vedaldi, A.: Fine-grained visual classification of aircraft. arXiv preprint arXiv:1306.5151 (2013)

[26] Murthy, S.K., Kasif, S., Salzberg, S.: Induction of oblique decision trees. In: The International Joint Conference on Artificial Intelligence. pp. 1002–1007 (1993)

[27] Perronnin, F., Sanchez, J., Mensink, T.: Improving the Fisher kernel for large-scale image classification. In: The European Conference on Computer Vision, LNCS, vol. 6314, pp. 143–156. Springer (2010)

[28] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 652–660 (2017)

[29] Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in neural information processing systems. pp. 5099–5108 (2017)

[30] Qin, T., Liu, T.: Introducing LETOR 4.0 datasets. CoRR **abs/1306.2597** (2013)

[31] Qin, T., Liu, T.Y., Li, H.: Query-level loss functions for information retrieval. Information Processing and Management **13**(4), 838–855 (2010)

[32] Qin, T., Zhang, X.D., Tsai, M.F., Wang, D.S., Liu, T.Y., Li, H.: Query-level loss functions for information retrieval. Information Processing and Management **2**(44), 838–855 (2008)

[33] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge. International Journal of Computer Vision **115**(3), 211–252 (2015)

[34] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: Inverted residuals and linear bottlenecks. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 4510–4520 (2018)

[35] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: The International Conference on Learning Representations. pp. 1–14 (2015)

[36] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. In: The International Conference on Machine Learning. pp. 1929–1959 (2014)

[37] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 2818–2826 (2016)

[38] Tanno, R., Arulkumaran, K., Alexander, D.C., Criminisi, A., Nori, A.: Adaptive neural trees. In: The International Conference on Machine Learning. pp. 6166–6175 (2019)

[39] Vijayakumar, S., Schaal, S.: An $O(n)$ algorithm for incremental real time learning in high dimensional space. In: The International Conference on Machine Learning. pp. 288–293 (2000)

[40] Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The Caltech-UCSD Birds-200-2011 Dataset. Tech. Rep. CNS-TR-2011-001, California Institute of Technology (2011)

[41] Wen, Z., Shi, J., He, B., Li, Q., Chen, J.: ThunderGBM: Fast GBDTs and random forests on GPUs. Website (2019), `https://github.com/Xtra-Computing/thundergbm`

[42] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3D ShapeNets: A deep representation for volumetric shapes. In: The IEEE Conference on Computer Vision and Pattern Recognition. pp. 1912–1920 (2015)

[43] Yu, H.T., Jatowt, A., Joho, H., Jose, J., Yang, X., Chen, L.: Wassrank: Listwise document ranking using optimal transport theory. In: The ACM International Conference on Web Search and Data Mining. pp. 24–32 (2019)

[44] Zhang, X., Jia, Y.: A linear discriminant analysis framework based on random subspace for face recognition. Pattern Recognition **40**, 2585–2591 (2007)

[45] Zhou, Z.H., Feng, J.: Deep forest: Towards an alternative to deep neural networks. In: The International Joint Conference on Artificial Intelligence. pp. 3553–3559 (2017)