IEICE TRANS. INF. & SYST., VOL.E105–D, NO.2 FEBRUARY 2022

PAPER

SimpleZSL: Extremely Simple and Fast Zero-Shot Learning with Nearest Neighbor Classifiers

Masayuki HIROMOTO[†a)], **Hisanao AKIMA**[†], ***Members***, **Teruo ISHIHARA**[†], ***and*** **Takuji YAMAMOTO**[†], ***Nonmembers***

SUMMARY Zero-shot learning (ZSL) aims to classify images of unseen classes by learning relationship between visual and semantic features. Existing works have been improving recognition accuracy from various approaches, but they employ computationally intensive algorithms that require iterative optimization. In this work, we revisit the primary approach of the pattern recognition, i.e., nearest neighbor classifiers, to solve the ZSL task by an extremely simple and fast way, called SimpleZSL. Our algorithm consists of the following three simple techniques: (1) just averaging feature vectors to obtain visual prototypes of seen classes, (2) calculating a pseudo-inverse matrix via singular value decomposition to generate visual features of unseen classes, and (3) inferring unseen classes by a nearest neighbor classifier in which cosine similarity is used to measure distance between feature vectors. Through the experiments on common datasets, the proposed method achieves good recognition accuracy with drastically small computational costs. The execution time of the proposed method on a single CPU is more than 100 times faster than those of the GPU implementations of the existing methods with comparable accuracies.

key words: *zero-shot learning, image recognition, nearest neighbor classifier, singular value decomposition*

1. Introduction

Image classification has been a representative problem to be solved by machine learning in decades. The most remarkable progress in recent years is the emergence of deep learning. Deep learning has greatly improved the classification accuracy for pattern recognition, and it is said that the deep learning now outperforms even humans [1], [2]. However, most of the works train their classifiers by supervised learning, which requires a large amount of labeled training data. On the other hand, humans can learn classes of objects just by seeing them few times or even without seeing. Learning from limited data is a next challenging problem for machine learning to realize more human-like AIs.

Few-shot learning (FSL) and zero-shot learning (ZSL) are the tasks that tackle the above problem. The more difficult one is ZSL, which aims to recognize objects whose instances have not been seen during training [3]–[8]. In ZSL image classification setting, semantic descriptions of both seen and unseen classes are provided in addition to the visual features of the seen classes. The task is to classify images of unseen classes by learning relationship between visual and semantic features.

Manuscript received April 21, 2021.
Manuscript revised August 24, 2021.
Manuscript publicized October 29, 2021.

[†]The authors are with Fujitsu Research, Fujitsu Limited, Kawasaki-shi, 211–8588 Japan.
a) E-mail: hiromoto@fujitsu.com

DOI: 10.1587/transinf.2021EDP7089

Existing works try to solve ZSL problem mainly from two approaches. One is a *non-generative* approach, in which visual-semantic relationship are learned by using embedding spaces. Some works learn projection functions from embedding spaces of visual features to those of semantic features, or functions in the opposite direction. There are also several works that utilize common intermediate embedding spaces to learn the visual-semantic relationship. In any case, such projection functions are obtained by optimization, where a loss function is minimized through iterative processes as in the case of the majority of the supervised learning methods. The other approach for ZSL is a *generative* one, which utilizes generative models to synthesize visual features of unseen classes from semantic descriptions. Generative adversarial networks (GAN) and variational auto-encoders (VAE) are often used as generators for visual features of unseen classes. These algorithms are powerful and they can generate plausible visual features useful for classification. However, their training process requires a large computational cost due to iterative optimization processes such as backpropagation.

In this paper, we try to solve the ZSL problem by an extremely simple way, named *SimpleZSL*. Our approach can be categorized into the generative one rather than the non-generative one, but it generates visual features by a very simple and deterministic way, more specifically, by calculating a pseudo-inverse matrix of semantic features. With this pseudo-inverse matrix, visual features of unseen classes are generated from visual prototypes of seen classes, which is obtained just by averaging visual features of training samples. Since this approach do not have any optimization process, the training of ZSL can be accelerated very much. In addition, when we use the visual features obtained by the above simple way, we found that a nearest neighbor classifier effectively works as a classifier for inference. The distance among the visual features is calculated by cosine similarity of feature vectors. This means that we do not need to train classifiers, which also shorten the training time of ZSL. Note that our work does not cover how to obtain visual and semantic features. We assume that they are given, as with other existing works. We do not take into account the computation for training of deep CNNs to extract visual features from images. For more information, please refer to experimental settings in Sect. 4.

Our contributions are summarized as follows:

- We propose SimpleZSL, which can generate visual features of unseen classes by a simple deterministic method without iterative optimization.
- Our method consists of the following simple techniques: (1) just averaging feature vectors to obtain visual prototypes of seen classes, (2) calculating a pseudo-inverse matrix via singular value decomposition to generate visual features of unseen classes, and (3) inferring unseen classes by a nearest neighbor classifier in which cosine similarity is used to measure distance between feature vectors.
- Experiments on common datasets show that SimpleZSL achieves good recognition accuracy with drastically small computational costs.

2. Related Work

Existing works for zero-shot learning can be categorized into two groups: non-generative and generative methods.

In the non-generative group, some works learns the mapping from visual features to semantic features to classify query samples of unseen classes in the semantic space. The earlier works [3], [9] adopt nearest neighbor classifiers, and the later works learn the mappings by using linear embeddings [10]–[14] or non-linear embeddings [15], [16] through optimization to minimize loss functions that are respectively defined in their works. On the other hand, there exits a reverse approach that learns the mapping from semantic to visual features [17]. There also exists an approach using bi-directional mapping between visual and semantic spaces [18], and some works that do not use the above mappings but learn the projection to a common intermediate space from both visual and semantic spaces [19]–[23] or directly learn classifiers without using mapping functions [24]. In any case, these methods learn projection functions from/to visual and semantic spaces via iterative optimization processes.

Another approach to ZSL is using generative models, which is a recent poplar approach and various works have been proposed [25]–[35] with the development of the generative models such as GANs and VAEs. These generative approaches generally outperform the non-generative models (see Sect. 4) at the cost of large computation needed to train the generative models.

In summary, the above existing ZSL methods include iterative optimization, resulting in long training time. Does it really needed for ZSL? Is there any easier way to solve ZSL problem? For this question, we found a hint in [36], which reports that a simple nearest neighbor classifier is still useful for few-shot learning (FSL). FSL is a task to learn classification function from a few training samples. Inspired by this work [36], in this paper we tackle ZSL by a very simple approach with a nearest neighbor classifier. Unlike the above mentioned non-generative earlier methods [3], [9], which utilize nearest neighbor classifiers for semantic features, our method takes a very simple generative approach and uses a nearest neighbor classifier for generated visual features. To the best of our knowledge, our work is the first generative ZSL method utilizing a nearest neighbor classifier for visual features. As reported in [17], classification on visual spaces also has an advantage to mitigate the effect of the hubness problem [37], which is a well-known issue for ZSL when utilizing a nearest neighbor classifier.

Note that there is another ZSL problem setting called transductive ZSL, different from the already explained setting called inductive ZSL. In the transductive setting, some unlabeled images from unseen classes are available during training. There are also several works for transductive ZSL. In this paper, however, we only consider the inductive setting and make the transductive problems out of scope.

3. Method

Our proposed method, called SimpleZSL, is based on the most primitive pattern recognition algorithm: nearest neighbor classifiers. In addition, SimpleZSL generates unseen visual features with a linear model by calculating a pseudo-inverse matrix of semantic features.

3.1 Definition and Notation

Here we define the ZSL problem. Suppose that we have a training sample set $\mathcal{D}_{\rm tr} = \{\boldsymbol{x}_i, y_i, \boldsymbol{a}_i\}_{i=1}^{N_{\rm tr}}$, which contains $N_{\rm tr}$ samples of visual features $\boldsymbol{x}_i \in \mathcal{X}_{\rm s}$, their corresponding class labels $y_i \in \mathcal{Y}_{\rm s}$, and semantic features $\boldsymbol{a}_i \in \mathcal{A}_{\rm s}$ of seen classes. The visual feature $\boldsymbol{x}_i$ is a $d_{\rm vis}$-dimension vector, which is obtained by a feature extractor (e.g., CNNs). The class labels y_i are one of the labels from a set of seen classes $\mathcal{Y}_{\rm s}$. The number of the seen classes is denoted by $n_{\rm sc} = |\mathcal{Y}_{\rm s}|$. The semantic feature $\boldsymbol{a}_i$ is a $d_{\rm att}$-dimension vector consisting of values between 0 to 1, which are the attribute values corresponding to the class y_i. Since the attributes are assigned for each class, the number of the possible semantic features is equal to the number of the seen classes, i.e., $|\mathcal{A}_{\rm s}| = |\mathcal{Y}_{\rm s}| = n_{\rm sc}$. A test set is denoted as $\mathcal{D}_{\rm ts} = \{\boldsymbol{x}_i, y_i, \boldsymbol{a}_i\}_{i=1}^{N_{\rm ts}}$, $\boldsymbol{x}_i \in \mathcal{X}_{\rm u}, y_i \in \mathcal{Y}_{\rm u}, \boldsymbol{a}_i \in \mathcal{A}_{\rm u}$, where $\mathcal{X}_{\rm u}$, $\mathcal{Y}_{\rm u}$, and $\mathcal{A}_{\rm u}$ are the sets of visual features, labels, and semantic features of unseen classes, respectively. The number of the unseen classes is denoted by $n_{\rm uc} = |\mathcal{Y}_{\rm u}| = |\mathcal{A}_{\rm u}|$. In ZSL, the semantic features of unseen classes, $\mathcal{A}_{\rm u}$, are available while training. The goal of ZSL is to learn a classifier to predict the labels $y_i \in \mathcal{Y}_{\rm u}$ for unseen visual features $\boldsymbol{x}_i \in \mathcal{X}_{\rm u}$, through training on $\mathcal{D}_{\rm tr}$ and $\mathcal{A}_{\rm u}$.

3.2 Overview

The training process of the proposed method is illustrated in Fig. 1. The training consists of two phases: the first phase to generate visual prototypes of seen classes (blue boxes in Fig. 1), and the second phase to generate visual features of unseen classes (yellow boxes in Fig. 1). As a result of the above two-phase training, visual prototypes for both seen and unseen classes are generated and stored in a prototype memory. The test process is realized by nearest neighbor

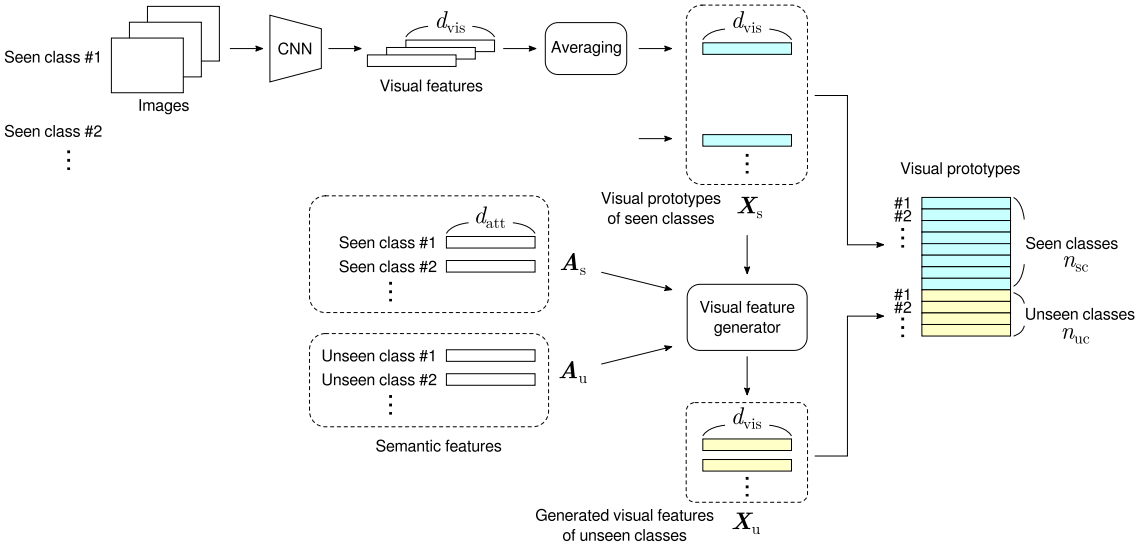


Fig. 1 A training process of SimpleZSL. The blue boxes represent visual prototypes of seen classes, which are calculated by averaging visual features of training samples. The yellow boxes are visual features of unseen classes, which are generated from semantic features of seen and unseen classes. See Fig. 2 for the calculation executed in the visual feature generator.

matching in the prototype memory. The detail of each process is explained below.

3.3 Prototype Generation for Seen Classes

In the first phase of the training process, visual prototypes of seen classes are generated just by averaging all the samples in a class. For each class c in seen classes $\mathcal{Y}_{\mathrm{s}}$, a visual prototype $\boldsymbol{x}_c^{\mathrm{proto}}$ is calculated as

$$\boldsymbol{x}_c^{\mathrm{proto}} = \frac{\sum_{\boldsymbol{x}_i \in \mathcal{X}_c} \boldsymbol{x}_i}{|\mathcal{X}_c|}, \quad c \in \mathcal{Y}_{\mathrm{s}}, \tag{1}$$

where $\mathcal{X}_c$ is a set of the visual features belonging to the class c. As a result, we obtain a series of n_{sc} visual prototypes of unseen classes, $\{\boldsymbol{x}_1^{\mathrm{proto}}, \cdots, \boldsymbol{x}_{n_{\mathrm{sc}}}^{\mathrm{proto}}\}$, which will be stored in the prototype memory.

3.4 Feature Generation for Unseen Classes

The second phase of the training process is the most important part of SimpleZSL. In this part, generated visual features of unseen classes $\{\boldsymbol{x}_c^{\mathrm{gen}}\}_{c \in \mathcal{Y}_{\mathrm{u}}}$ are calculated from the visual prototypes of seen classes $\{\boldsymbol{x}_c^{\mathrm{proto}}\}_{c \in \mathcal{Y}_{\mathrm{s}}}$ with the help of the semantic features of both seen classes $\mathcal{A}_{\mathrm{s}}$ and unseen classes $\mathcal{A}_{\mathrm{u}}$. Let us represent the above visual and semantic features of seen and unseen classes in matrix forms as follows:

$$\{\boldsymbol{x}_c^{\mathrm{proto}}\}_{c \in \mathcal{Y}_{\mathrm{s}}} := \boldsymbol{X}_{\mathrm{s}} \in \mathbb{R}^{n_{\mathrm{sc}} \times d_{\mathrm{vis}}}, \tag{2}$$

$$\{\boldsymbol{x}_c^{\mathrm{gen}}\}_{c \in \mathcal{Y}_{\mathrm{u}}} := \boldsymbol{X}_{\mathrm{u}} \in \mathbb{R}^{n_{\mathrm{uc}} \times d_{\mathrm{vis}}}, \tag{3}$$

$$\mathcal{A}_{\mathrm{s}} := \boldsymbol{A}_{\mathrm{s}} \in \mathbb{R}^{n_{\mathrm{sc}} \times d_{\mathrm{att}}}, \tag{4}$$

$$\mathcal{A}_{\mathrm{u}} := \boldsymbol{A}_{\mathrm{u}} \in \mathbb{R}^{n_{\mathrm{uc}} \times d_{\mathrm{att}}}. \tag{5}$$

What we have to do is first learning a projection function $f(\cdot)$ that relates visual and semantic spaces as

$$\boldsymbol{A}_{\mathrm{s}} = f(\boldsymbol{X}_{\mathrm{s}}), \tag{6}$$

and then generating unseen visual features by

$$\boldsymbol{X}_{\mathrm{u}} = f^{-1}(\boldsymbol{A}_{\mathrm{u}}). \tag{7}$$

In this work, we adopt a linear model for f. With a projection matrix $\boldsymbol{W} \in \mathbb{R}^{d_{\mathrm{att}} \times d_{\mathrm{vis}}}$, Eqs. (6) and (7) are written as

$$\boldsymbol{X}_{\mathrm{s}} = \boldsymbol{A}_{\mathrm{s}}\boldsymbol{W}, \tag{8}$$

$$\boldsymbol{X}_{\mathrm{u}} = \boldsymbol{A}_{\mathrm{u}}\boldsymbol{W} = \boldsymbol{A}_{\mathrm{u}}\boldsymbol{A}_{\mathrm{s}}^{-1}\boldsymbol{X}_{\mathrm{s}}. \tag{9}$$

Note that each row of the projection matrix $\boldsymbol{W}$ represents a decomposed visual feature corresponding to a particular entry of the semantic feature. $\boldsymbol{W}$ can be calculated by $\boldsymbol{A}_{\mathrm{s}}^{-1}\boldsymbol{X}_{\mathrm{s}}$. Since the inverse matrix of $\boldsymbol{A}_{\mathrm{s}}$ is not possible to calculate directly, we utilize a pseudo-inverse matrix $\boldsymbol{A}_{\mathrm{s}}^{+}$ instead of $\boldsymbol{A}_{\mathrm{s}}^{-1}$. The procedure to generate unseen visual features becomes

$$\boldsymbol{X}_{\mathrm{u}} = \boldsymbol{A}_{\mathrm{u}}\boldsymbol{A}_{\mathrm{s}}^{+}\boldsymbol{X}_{\mathrm{s}}, \tag{10}$$

as illustrated in Fig. 2.

In our method, a pseudo-inverse matrix is calculated by using singular value decomposition (SVD). If

$$\boldsymbol{A} = \boldsymbol{Q}_1\boldsymbol{\Sigma}\boldsymbol{Q}_2^{\mathrm{T}} \tag{11}$$

is the singular value decomposition of $\boldsymbol{A}$, we have the pseudo-inverse matrix

$$\boldsymbol{A}^{+} = \boldsymbol{Q}_2\boldsymbol{\Sigma}^{+}\boldsymbol{Q}_1^{\mathrm{T}}, \tag{12}$$

where $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ are orthogonal matrices, $\boldsymbol{\Sigma}$ is a diagonal matrix consisting of $\boldsymbol{A}$'s large singular values (followed by

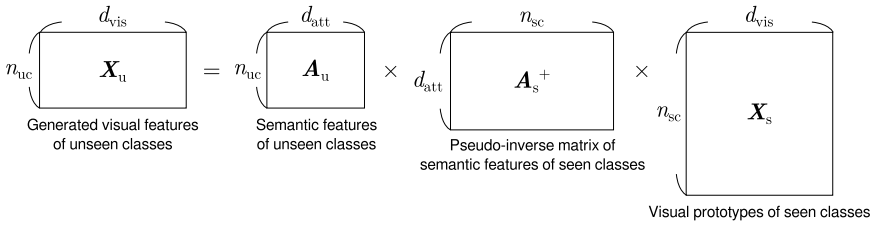


Fig. 2 The calculation for feature generation in the proposed method. n_{sc} and n_{uc} are the number of seen and unseen classes, and d_{vis} and d_{att} are the dimensions of the visual and semantic feature vectors, respectively.

zeros), and $\boldsymbol{\Sigma}^{+}$ is the diagonal matrix consisting of the reciprocals of $\boldsymbol{A}$'s large singular values (again, followed by zeros). As observed in Sect. 4.5.1, the number of the large singular values in $\boldsymbol{\Sigma}$ affects the classification accuracy of ZSL. In our method, up to n-th largest singular values of $\boldsymbol{A}_{\mathrm{s}}$ are included and the others are replaced with zeros. n is determined by a cutoff parameter α as

$$n = \mathrm{int}(\alpha N), \tag{13}$$

where N is the size of $\boldsymbol{\Sigma}$, in concrete, which is equal to the smaller value of d_{att} and n_{sc}.

Although the computational complexity of SVD is $O(N^3)$, this does not matter because $N = \min(d_{\mathrm{att}}, n_{\mathrm{sc}})$ is not a large value (usually tens or hundreds). As a result, SimpleZSL realizes fast generation of unseen visual features without heavy optimization processes.

3.5 Nearest Neighbor Classifier

For a test, SimpleZSL utilizes a nearest neighbor classifier to predict labels for unseen images. Given the visual prototypes of unseen classes obtained in the training process, our classifier selects a class label c whose visual prototype is most similar to the query $\boldsymbol{x}$ by a nearest neighbor rule:

$$y(\boldsymbol{x}) = \underset{c \in \mathcal{Y}_{\mathrm{u}}}{\arg\max}\ \mathrm{sim}(\boldsymbol{x}, \boldsymbol{x}_c^{\mathrm{gen}}), \tag{14}$$

where sim(,) measures cosine similarity of two vectors, which is defined as

$$\mathrm{sim}(\boldsymbol{v}_1, \boldsymbol{v}_2) = \cos(\boldsymbol{v}_1, \boldsymbol{v}_2) = \frac{\boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{||\boldsymbol{v}_1||||\boldsymbol{v}_1||}. \tag{15}$$

3.6 Generalized Zero-Shot Learning

SimpleZSL can be easily extended to generalized ZSL (GZSL) settings. In GZSL, a test query is selected from both seen and unseen classes, and the classifier is required to predict the label from a union set of seen and unseen classes. We can solve GZSL problem with SimpleZSL just by slightly modifying the nearest neighbor classifier as follows:

$$y(\boldsymbol{x}) = \underset{c \in \mathcal{Y}_{\mathrm{s}} \cup \mathcal{Y}_{\mathrm{u}}}{\arg\max}\ \mathrm{sim}(\boldsymbol{x}, \boldsymbol{x}_c^{\mathrm{proto/gen}}). \tag{16}$$

This means that the query $\boldsymbol{x}$ is matched not only with unseen classes but also with seen classes. However, it often happens that a query is more likely to be classified into seen classes than unseen classes. Although we can not simultaneously improve classification accuracy for both seen and unseen classes because there is a trade-off between them, we can increase their harmonic mean (often used to measure GZSL performance) by adjusting a parameter affecting the trade-off. In this work, we introduce a offset parameter β that assigns weights for similarity measurement depending on whether the visual prototype belongs to seen or unseen classes:

$$\mathrm{sim}(\boldsymbol{x}, \boldsymbol{x}_c^{\mathrm{proto/gen}}) = \begin{cases} \cos(\boldsymbol{x}, \boldsymbol{x}_c^{\mathrm{proto}}), & c \in \mathcal{Y}_{\mathrm{s}} \\ \cos(\boldsymbol{x}, \boldsymbol{x}_c^{\mathrm{gen}}) + \beta, & c \in \mathcal{Y}_{\mathrm{u}} \end{cases}. \tag{17}$$

Note that, since GZSL is a difficult problem as discussed in other works [29], [40], [42], our method is not a fundamental solution for GZSL. However, we found that our method effectively works for several datasets as shown in the experimental results.

4. Experiments

4.1 Dataset and Evaluation Criteria

The performance of the proposed method is evaluated on widely-used datasets: SUN [38], CUB [39], AWA1 [3], AWA2 [40], and aPY [41]. The statistics of the datasets are summarized in Table 1. For the evaluation criteria, we follow the GUB setting [40]. The classification accuracy is measured with Top-1 (T1) accuracy. In the case of GZSL, it is measured with the harmonic mean of accuracies, $H = 2US/(U+S)$, where U and S are accuracies of unseen and seen classes, respectively.

4.2 Implementation

The visual features are extracted by ResNet101 [1], as in GUB [40]. The dimension of the visual feature is $d_{\mathrm{vis}} = 2048$. Following [36], we apply two types of preprocessing for visual features, normalizing and centering. We perform both normalizing and centering for CUB and SUN datasets, and only centering for the other datasets. We directly use the attributes whose values are between 0 to 1 as semantic

Table 1 Statistics of the datasets used for evaluation, in terms of the number attributes (d_{att}), the number of seen (n_{sc}) and unseen (n_{uc}) classes, and the number of samples for training ($\mathcal{D}_{\text{tr}}$) and test ($\mathcal{D}_{\text{ts}}$). The breakdown of the number of seen and unseen samples in the test set is also shown.

Dataset	d_{att}	n_{sc}	n_{uc}	$\mathcal{D}_{\text{tr}}$	$\mathcal{D}_{\text{ts}}$ (seen)	$\mathcal{D}_{\text{ts}}$ (unseen)
SUN [38]	102	645	72	10320	2580	1440
CUB [39]	312	150	50	7057	1764	2967
AWA1 [3]	85	40	10	19832	4958	5685
AWA2 [40]	85	40	10	23527	5882	7913
aPY [41]	64	20	12	5932	1483	7924

Table 2 Zero-shot learning accuracy on SUN, CUB, AWA1/2, and aPY datasets. T1 means Top-1 accuracy of ZSL. U, S, and H are accuracies of GZSL, where U and S mean unseen and unseen classes and H is their harmonic mean. Methods in upper rows are non-generative, and those in lower rows are generative ones. The best result in each category is in **bold**. All the values are in %. '*' indicates re-implementation by [40] to follow GUB settings.

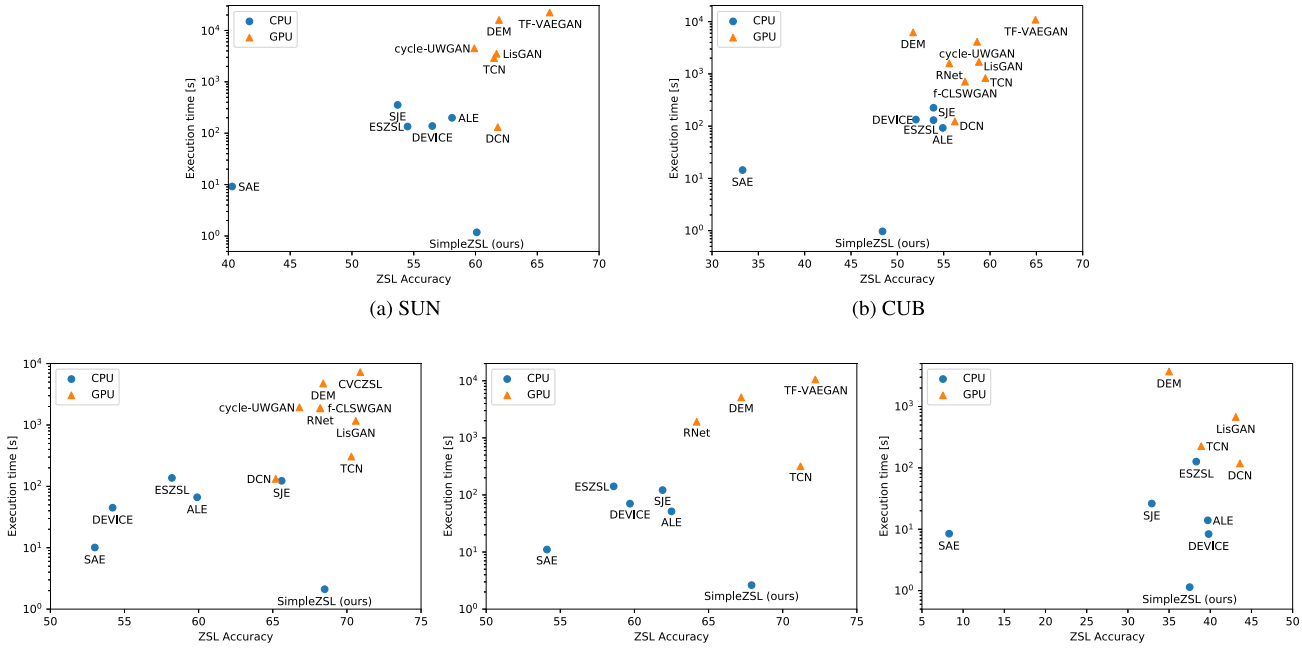
	SUN				CUB				AWA1				AWA2				aPY			
	ZSL	GZSL			ZSL	GZSL			ZSL	GZSL			ZSL	GZSL			ZSL	GZSL		
Method	T1	U	S	H	T1	U	S	H	T1	U	S	H	T1	U	S	H	T1	U	S	H
DAP* [3]	39.9	4.2	25.1	7.2	40.0	1.7	67.9	3.3	44.1	0.0	88.7	0.0	46.1	0.0	84.7	0.0	33.8	4.8	78.3	9.0
IAP* [3]	19.4	1.0	37.8	1.8	24.0	0.2	**72.8**	0.4	35.9	2.1	78.2	4.1	35.9	0.9	87.6	1.8	36.6	5.7	65.6	10.4
CONSE* [9]	38.8	6.8	39.9	11.6	34.3	0.6	72.2	3.1	45.6	0.4	88.6	0.8	44.5	0.5	90.6	1.0	26.9	0.0	**91.2**	0.0
CMT* [16]	39.9	8.1	21.8	11.8	34.6	7.2	49.8	12.6	39.5	0.9	87.6	1.8	37.9	0.5	90.0	1.0	28.0	1.4	85.2	2.8
DEVISE* [11]	56.5	16.9	27.4	20.9	52.0	23.8	53.0	32.8	54.2	13.4	68.7	22.4	59.7	17.1	74.7	27.8	39.8	4.9	76.9	9.2
SSE* [19]	51.5	2.1	36.4	4.0	43.9	8.5	46.9	14.4	60.1	7.0	80.5	12.9	61.0	8.1	82.5	14.8	34.0	0.2	78.9	0.4
ALE* [10]	58.1	21.8	33.1	26.3	54.9	23.7	62.8	34.4	59.9	16.8	76.1	27.5	62.5	14.0	81.8	23.9	39.7	4.6	73.7	8.7
SJE* [12]	53.7	14.7	30.5	19.8	53.9	23.5	59.2	33.6	65.6	11.3	74.6	19.6	61.9	8.0	73.9	14.4	32.9	3.7	55.7	6.9
ESZSL* [13]	54.5	11.0	27.9	15.8	53.9	12.6	63.8	21.0	58.2	6.6	75.6	12.1	58.6	5.9	77.8	11.0	38.3	2.4	70.7	4.6
LATEM* [15]	55.3	14.7	28.8	19.5	49.3	15.2	57.3	24.0	55.1	7.3	71.7	13.3	55.8	11.5	77.3	20.0	35.2	0.1	73.0	0.2
SYNC* [20]	56.3	7.9	**43.3**	13.4	55.6	11.5	70.9	19.8	54.0	8.9	87.3	16.2	46.6	10.0	90.5	18.0	23.9	7.4	66.3	13.3
SAE* [14]	40.3	8.8	18.0	11.8	33.3	7.8	54.0	13.6	53.0	1.8	77.1	3.5	54.1	1.1	82.2	2.2	8.3	0.4	80.9	0.9
DEM [17]	61.9	20.5	34.3	25.6	51.7	19.6	57.9	29.2	68.4	32.8	84.7	47.3	67.2	30.5	86.4	45.1	35.0	11.1	75.1	19.4
RNet [21]	-	-	-	-	55.6	38.1	61.1	47.0	68.2	31.4	**91.3**	46.7	64.2	30.0	**93.4**	45.3	-	-	-	-
PSR [22]	61.4	20.8	37.2	26.7	56.0	24.6	54.3	33.9	-	-	-	-	63.8	20.7	73.8	32.3	38.4	13.5	51.4	21.4
DCN [23]	61.8	25.5	37.0	30.2	**56.2**	28.4	60.7	38.7	65.2	25.5	84.2	39.1	-	-	-	-	43.6	14.2	75.0	23.9
ZSKL [18]	61.7	20.1	31.4	24.5	51.7	21.6	52.8	30.6	70.1	17.9	82.2	29.4	70.5	18.9	82.7	30.8	**45.3**	10.5	76.2	18.5
CVCZSL [24]	**62.6**	**36.3**	42.8	**39.3**	54.4	**47.4**	47.6	**47.5**	**70.9**	**62.7**	77.0	**69.1**	**71.1**	**56.4**	81.4	**66.7**	38.0	**26.5**	74.0	**39.0**
GFZSL* [25]	60.6	0.0	39.6	0.0	49.3	0.0	45.7	0.0	68.3	1.8	80.3	3.5	63.8	2.5	80.1	4.8	38.4	0.0	**83.3**	0.0
SP-AEN [26]	59.2	24.9	38.6	30.3	55.4	34.7	**70.6**	46.6	-	-	-	-	58.5	23.3	**90.9**	37.1	24.1	13.7	63.4	22.6
cycle-UWGAN [27]	59.9	47.2	33.8	39.4	58.6	47.9	59.3	53.0	66.8	59.6	63.4	59.8	-	-	-	-	-	-	-	-
f-CLSWGAN [28]	60.8	42.6	36.6	39.4	57.3	57.7	43.7	49.7	68.2	57.9	61.4	59.6	-	-	-	-	-	-	-	-
SE-(G)ZSL [29]	63.4	40.9	30.5	34.9	59.6	41.5	53.3	46.7	69.5	56.3	67.8	61.5	69.2	58.3	68.1	62.8	-	-	-	-
CADA-VAE [30]	-	47.2	35.7	40.6	-	51.6	53.5	52.4	-	57.3	72.8	64.1	-	55.8	75.0	63.9	-	-	-	-
LisGAN [31]	61.7	42.9	37.8	40.2	58.8	46.5	57.9	51.6	70.6	52.6	76.3	62.3	-	-	-	-	43.1	-	-	-
TCN [32]	61.5	31.2	37.3	34.0	59.5	52.6	52.0	52.3	70.3	49.4	76.5	60.0	71.2	**61.2**	65.8	63.4	38.9	24.1	64.0	35.1
f-VAEGAN [33]	64.7	45.1	38.0	41.3	61.0	48.4	60.1	53.6	-	-	-	-	71.1	57.6	70.6	63.5	-	-	-	-
TF-VAEGAN [34]	66.0	45.6	40.7	43.0	64.9	52.8	64.7	58.1	-	-	-	-	72.2	59.8	75.1	66.6	-	-	-	-
IZF-Softmax [35]	**68.4**	**52.7**	**57.0**	**54.8**	**67.1**	**52.7**	68.0	**59.4**	**74.3**	**61.3**	**80.5**	**69.6**	**74.5**	60.6	77.5	**68.0**	**44.9**	**42.3**	60.5	**49.8**
SimpleZSL (ours)	60.1	52.8	29.9	38.2	48.4	44.2	36.0	39.7	68.5	56.7	75.5	64.8	67.9	52.5	79.7	63.3	37.5	29.7	63.4	40.4

features. The program is written in Python scripting language. We use NumPy library to implement SVD to compute a pseudo-inverse matrix.

4.3 Comparison with State-of-the-Art Methods

The performance of the proposed method and other ZSL methods including state-of-the-art ones is compared in Table 2. For SimpleZSL, the cutoff parameter for SVD is set to $\alpha = 0.7$ and the offset parameter for GZSL is set to $\beta = 0.13$. As shown in the results, ZSL accuracy of SimpleZSL is worse than the state-of-the-art methods, but better than most of the non-generative methods and comparable to the earlier generative ones. For GZSL, H scores, which indicate the GZSL performance considering the trade-off between accuracies for seen and unseen classes, show similar results as ZSL accuracy. We can see that SimpleZSL achieves good performance in spite of its extremely simple algorithm.

The reason why our SimpleZSL shows lower performance than some latest algorithms can be simply considered as follows. In ZSL algorithms with generative approach, a function that generates visual features from semantic features is learned. We utilize a very simple linear function for this purpose, whereas other methods adopt GAN-based complex and expressive functions at the cost of large computations. Our result indicates the potential of the simple linear generative function, and at the same time its limitation for recognition performance compared to more com-



(a) SUN (b) CUB

(c) AWA1 (d) AWA2 (e) aPY

Fig. 3 Measured execution time of ZSL algorithms against their accuracy. The methods marked as circles are run on a CPU, while the triangles also use a GPU. Our SimpleZSL is remarkably faster than the existing methods.

Table 3 Deep learning frameworks and source code URLs used to measure execution time of ZSL algorithms.

Method	DL framework	URL
ALE [10], DeViSE [11], SJE [12], ESZSL [13], SAE [14]	n/a	https://github.com/mvp18/Popular-ZSL-Algorithms
DEM [17]	TensorFlow 1.15	https://github.com/lzrobots/DeepEmbeddingModel_ZSL
RNet [21]	PyTorch 0.4.1	https://github.com/lzrobots/LearningToCompare_ZSL
DCN [23]	PyTorch 0.4.1	https://github.com/Nir3usHaHaHa/Generalized-Zero-Shot-Learning-with-Deep-Calibration-Network-NIPS18
CVCZSL [24]	PyTorch 0.4.1	https://github.com/kailigo/cvcZSL
f-CLSWGAN [28]	PyTorch 1.7.1	https://github.com/JieRen98/f-CLSWGAN
LisGAN [31]	PyTorch 0.4.1	https://github.com/lijin118/LisGAN
TCN [32]	PyTorch 0.4.1	http://vipl.ict.ac.cn/resources/codes/code/TCN_ICCV2019.zip
TF-VAEGAN [34]	PyTorch 0.3.1	https://github.com/akshitac8/tfvaegan

plex functions.

4.4 Computational Cost

To see the efficiency of our algorithm, we measure the execution time of the ZSL algorithms. Figure 3 plots the measured execution time against the reported ZSL accuracy of the existing works and our proposed method. We use publicly available source codes as listed in Table 3 and follow default parameter settings for each dataset. We use GPU implementation if available (plotted by triangles in Fig. 3). Our SimpleZSL and some earlier methods are run on a CPU without using a GPU (circles in Fig. 3). The machine specs used for the evaluation are summarized in Table 4. Note that the measured execution time of all the methods is the total of training and testing processes on each datset, but does not include the visual feature extraction process by CNNs. We could not evaluate all the methods listed in Table 2 since some works do not publish their source codes or the source codes could not be executed in our environment because of their dependencies on old libraries currently not available.

Table 4 Specifications of the evaluation machine.

CPU	Intel Core i7-9700K @ 3.60 GHz
RAM	16 GB
OS	Linux 5.10
GPU	NVIDIA GeForce RTX 2070
VRAM	8 GB
CUDA	11.2

For the existing methods, we can see that the execution time exponentially increases as accuracy improves, although the complex methods employ GPU acceleration. The proposed method, however, realizes drastically fast execution in less than three seconds. This is more than 100 times

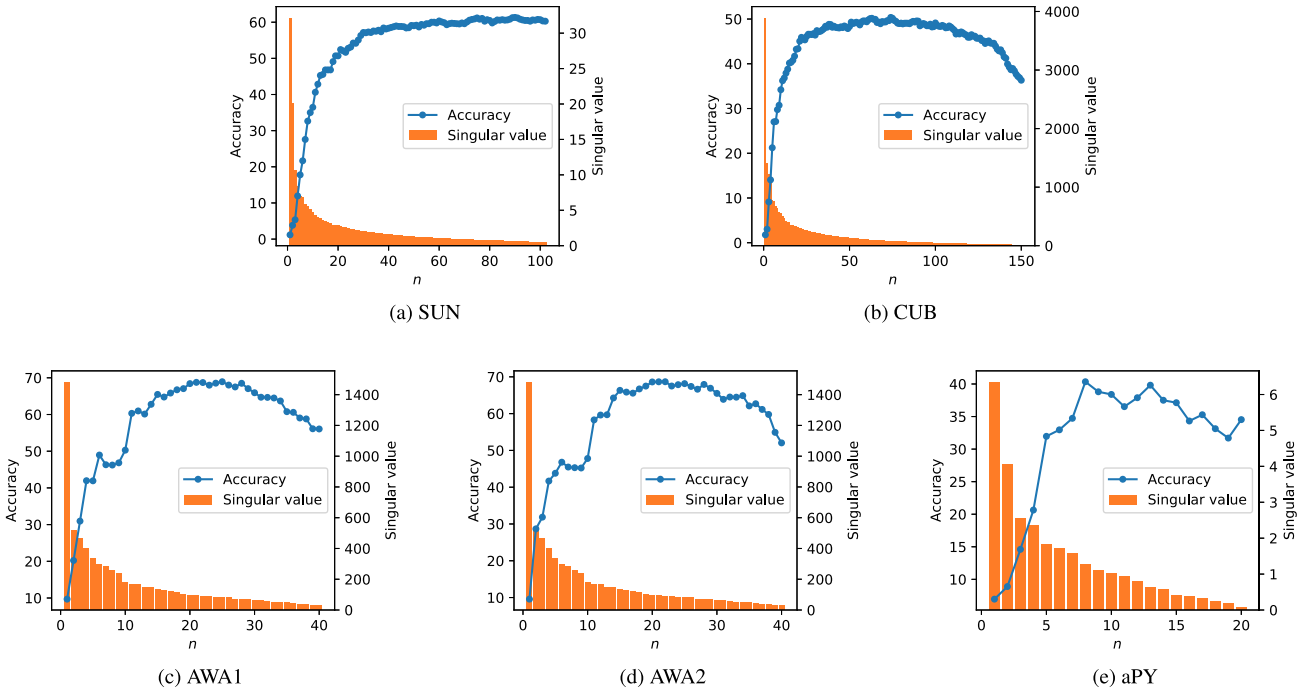


Fig. 4 Change of ZSL accuracy when the cutoff parameter α is changed. Accuracy is shown in %. The x-axis is plotted with $n = \mathrm{int}(\alpha N)$, where N is the number of the singular values. The magnitude of singular values is also plotted, where n represents the index of the singular values in descending order.

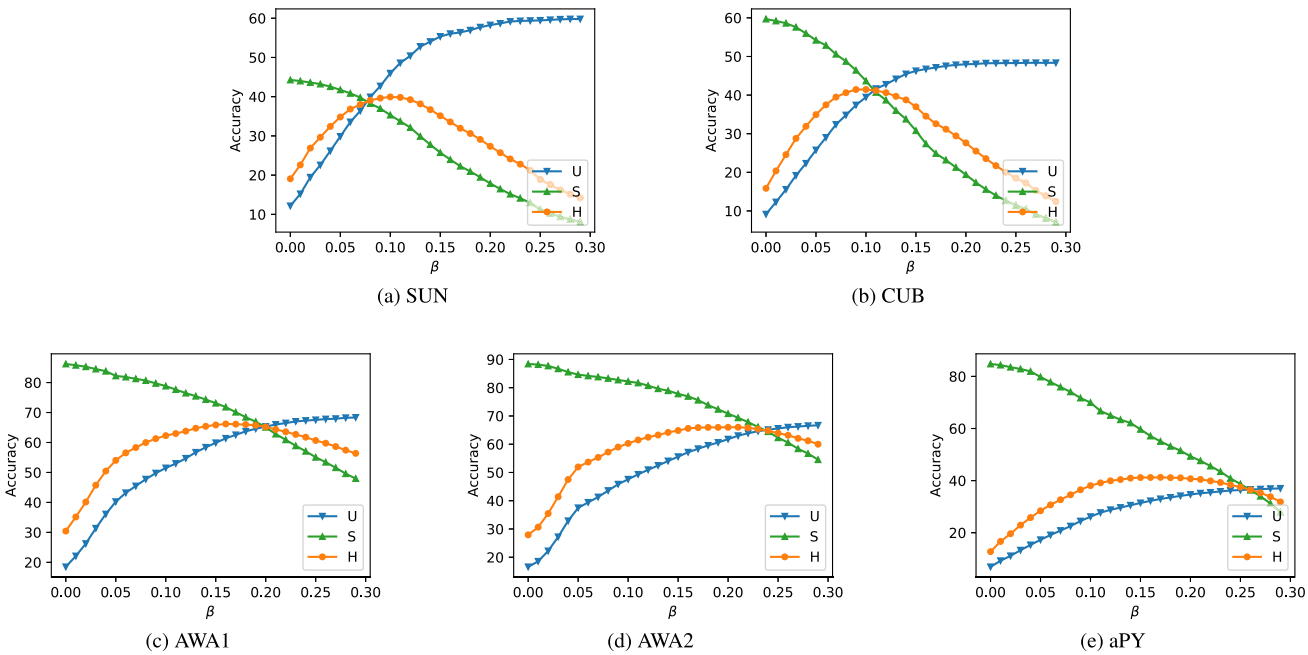


Fig. 5 Change of GZSL performance when the offset parameter β is changed. U and S are accuracies of unseen and seen classes, and H is their harmonic mean. The values are in %.

faster than those of the GPU implementations of the existing methods with comparable accuracies under SUN, CUB, and AWA1/2 datasets. The speedup for aPY dataset is smaller than the other datasets because of its small scale of the problem, but still more than 7x faster than the other methods. This result shows that SimpleZSL is suitable for resource-limited computing environments, such as mobile or embedded devices.

4.5 Ablation Study

In the proposed method, two parameters α and β affecting ZSL and GZSL performance are utilized. We conduct further analysis to see the effect of these parameters.

4.5.1 Cutoff Parameter for SVD

As described in Sect. 3.4, the cutoff parameter α determines how many singular values are used to calculate a pseudo-inverse matrix. Figure 4 shows the change of T1 accuracy of ZSL when α is swept from 0 to 1. The magnitude of the singular values is also plotted in descending order. We can see that the accuracy is improved as α increases from 0. However, in the region near $\alpha = 1$, the accuracy drops except for SUN dataset. These results imply the following two facts. One is that the larger singular values are important for feature generation. It is considered that the singular values behave as latent variables for semantic features. The other is that utilization of all the small singular values spoils the quality of feature generation. Too many singular values may cause a similar situation to so-called over fitting. Through this analysis, we manually determine the cutoff parameter as $\alpha = 0.7$, performing well for all the datasets.

4.5.2 Offset Parameter for GZSL

Our method increases the value of the harmonic mean H, which is used to measure GZSL performance, by adding an offset β to the similarity scores of unseen classes as described in Eq. (17). To see how the offset parameter affects the classification accuracy of seen and unseen classes, we perform additional experiments by changing the parameter β. Figure 5 shows the results on the five datasets. We can see trade-offs between the accuracies of seen (S) and unseen (U) classes, as mentioned in Sect. 3.6. As a result, their harmonic mean H has a peak around from $\beta = 0.1$ to 0.2, in general.

As we can see from the results, the best offset parameters depend on datasets. If you want to determine the best offset for each dataset, you can use validation data from the training set to search an optimal parameter. The required execution time for grid search of β in 30 steps like Fig. 5 is 1 or 2 seconds, which does not spoil the fast execution of our SimpleZSL. However, particularly for AWA1/2 and aPY datasets, because of the small number of the seen classes, separation of the validation set from the training set may deteriorate classification accuracy. Therefore, we do not use validation set but manually determines the parameter as $\beta = 0.13$ for the experiment in Sect. 4.3. Note that the utilization of the common parameter $\beta = 0.13$ through all the datasets is a severe condition, while some works tune hyperparameters for each dataset.

5. Conclusion

In this paper, we propose SimpleZSL, which consists of very simple techniques—just averaging feature vectors to obtain visual prototypes of seen classes, calculating a pseudo-inverse matrix via singular value decomposition to generate visual features of unseen classes, and inferring unseen classes by a nearest neighbor classifier. These calculations do not include iterative optimization that incurs large computational costs. Through the experiments, SimpleZSL shows good recognition accuracy and achieves more than 100 times faster than the existing methods with comparable accuracies. This work gives us the findings that a simple linear model is sufficient to generate visual features of unseen classes and that the nearest neighbor classifier is still useful even for ZSL, which is one of the recent challenging problems in computer vision. We hope that this work will be an opportunity to rethink the approach for ZSL, and also will be used as a baseline to compare the performance of future ZSL algorithms.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," IEEE Conf. Comput. Vis. Pattern Recog., pp.770–778, 2016.
[2] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," IEEE Conf. Comput. Vis. Pattern Recog., pp.7132–7141, 2018.
[3] C.H. Lampert, H. Nickisch, and S. Harmeling, "Attribute-based classification for zero-shot visual object categorization," IEEE Trans. Pattern Anal. Mach. Intell., vol.36, no.3, pp.453–465, 2013.
[4] H. Larochelle, D. Erhan, and Y. Bengio, "Zero-data learning of new tasks," AAAI, vol.2, pp.646–651, 2008.
[5] M. Rohrbach, M. Stark, and B. Schiele, "Evaluating knowledge transfer and zero-shot learning in a large-scale setting," IEEE Conf. Comput. Vis. Pattern Recog., pp.1641–1648, 2011.
[6] X. Yu and Y. Aloimonos, "Attribute-based transfer learning for object categorization with zero/one training example," Eur. Conf. Comput. Vis., pp.127–140, 2010.
[7] X. Xu, F. Shen, Y. Yang, D. Zhang, H. Tao Shen, and J. Song, "Matrix tri-factorization with manifold regularizations for zero-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.2007–2016, 2017.
[8] Z. Ding, M. Shao, and Y. Fu, "Low-rank embedded ensemble semantic dictionary for zero-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.2050–2058, 2017.
[9] M. Norouzi, T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G.S. Corrado, and J. Dean, "Zero-shot learning by convex combination of semantic embeddings," Int. Conf. Learn. Represent., 2014.
[10] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid, "Label-embedding for image classification," IEEE Trans. Pattern Anal. Mach. Intell., vol.38, no.7, pp.1425–1438, 2016.
[11] A. Frome, G.S. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov, "Devise: A deep visual-semantic embedding model," Adv. Neural Inform. Process. Syst., pp.2121–2129, 2013.
[12] Z. Akata, S. Reed, D. Walter, H. Lee, and B. Schiele, "Evaluation of output embeddings for fine-grained image classification," IEEE Conf. Comput. Vis. Pattern Recog., pp.2927–2936, 2015.
[13] B. Romera-Paredes and P. Torr, "An embarrassingly simple approach to zero-shot learning," Int. Conf. Mach. Learn., pp.2152–2161, 2015.
[14] E. Kodirov, T. Xiang, and S. Gong, "Semantic autoencoder for zero-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.4447–4456, 2017.
[15] Y. Xian, Z. Akata, G. Sharma, Q. Nguyen, M. Hein, and B. Schiele, "Latent embeddings for zero-shot classification," IEEE Conf. Comput. Vis. Pattern Recog., pp.69–77, 2016.
[16] R. Socher, M. Ganjoo, C.D. Manning, and A. Ng, "Zero-shot learning through cross-modal transfer," Adv. Neural Inform. Process. Syst., pp.935–943, 2013.
[17] L. Zhang, T. Xiang, and S. Gong, "Learning a deep embedding

model for zero-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.2021–2030, 2017.
[18] H. Zhang and P. Koniusz, "Zero-shot kernel learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.7670–7679, 2018.
[19] Z. Zhang and V. Saligrama, "Zero-shot learning via semantic similarity embedding," Int. Conf. Comput. Vis., pp.4166–4174, 2015.
[20] S. Changpinyo, W.-L. Chao, B. Gong, and F. Sha, "Synthesized classifiers for zero-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.5327–5336, 2016.
[21] F. Sung, Y. Yang, L. Zhang, T. Xiang, P.H.S. Torr, and T.M. Hospedales, "Learning to compare: Relation network for few-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.1199–1208, 2018.
[22] Y. Annadani and S. Biswas, "Preserving semantic relations for zero-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.7603–7612, 2018.
[23] S. Liu, M. Long, J. Wang, and M.I. Jordan, "Generalized zero-shot learning with deep calibration network," Adv. Neural Inform. Process. Syst., pp.2005–2015, 2018.
[24] K. Li, M.R. Min, and Y. Fu, "Rethinking zero-shot learning: A conditional visual classification perspective," Int. Conf. Comput. Vis., pp.3583–3592, 2019.
[25] V.K. Verma and P. Rai, "A simple exponential family framework for zero-shot learning," Eur. Conf. Mach. Learn. Princ. Pract. Knowl. Discov., vol.10535, pp.792–808, 2017.
[26] L. Chen, H. Zhang, J. Xiao, W. Liu, and S.F. Chang, "Zero-shot visual recognition using semantics-preserving adversarial embedding networks," IEEE Conf. Comput. Vis. Pattern Recog., pp.1043–1052, 2018.
[27] R. Felix, V.B. Kumar, I. Reid, and G. Carneiro, "Multi-modal cycle-consistent generalized zero-shot learning," Eur. Conf. Comput. Vis., vol.11210, pp.21–37, 2018.
[28] Y. Xian, T. Lorenz, B. Schiele, and Z. Akata, "Feature generating networks for zero-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.5542–5551, 2018.
[29] V. Kumar Verma, G. Arora, A. Mishra, and P. Rai, "Generalized zero-shot learning via synthesized examples," IEEE Conf. Comput. Vis. Pattern Recog., pp.4281–4289, 2018.
[30] E. Schonfeld, S. Ebrahimi, S. Sinha, T. Darrell, and Z. Akata, "Generalized zero-and few-shot learning via aligned variational autoencoders," IEEE Conf. Comput. Vis. Pattern Recog., pp.8239–8247, 2019.
[31] J. Li, M. Jing, K. Lu, Z. Ding, L. Zhu, and Z. Huang, "Leveraging the invariant side of generative zero-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.7394–7403, 2019.
[32] H. Jiang, R. Wang, S. Shan, and X. Chen, "Transferable contrastive network for generalized zero-shot learning," Int. Conf. Comput. Vis., pp.9764–9773, 2019.
[33] Y. Xian, S. Sharma, B. Schiele, and Z. Akata, "f-VAEGAN-D2: A feature generating framework for any-shot learning," IEEE Conf. Comput. Vis. Pattern Recog., pp.10267–10276, 2019.
[34] S. Narayan, A. Gupta, F.S. Khan, C.G.M. Snoek, and L. Shao, "Latent embedding feedback and discriminative features for zero-shot classification," Eur. Conf. Comput. Vis., pp.479–495, 2020.
[35] Y. Shen, J. Qin, L. Huang, L. Liu, F. Zhu, and L. Shao, "Invertible zero-shot recognition flows," Eur. Conf. Comput. Vis., vol.12361, pp.614–631, 2020.
[36] Y. Wang, W.L. Chao, K.Q. Weinberger, and L. van der Maaten, "SimpleShot: Revisiting nearest-neighbor classification for few-shot learning," arXiv:1911.04623, 2019.
[37] A. Lazaridou, G. Dinu, and M. Baroni, "Hubness and pollution: Delving into cross-space mapping for zero-shot learning," ACL-IJCNLP, pp.270–280, 2015.
[38] G. Patterson and J. Hays, "SUN attribute database: Discovering, annotating, and recognizing scene attributes," IEEE Conf. Comput. Vis. Pattern Recog., pp.2751–2758, 2012.
[39] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD Birds 200," tech. rep., California Institute of Technology, 2010.
[40] Y. Xian, C.H. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly," IEEE Trans. Pattern Anal. Mach. Intell., vol.41, no.9, pp.2251–2265, 2018.
[41] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth, "Describing objects by their attributes," IEEE Conf. Comput. Vis. Pattern Recog., pp.1778–1785, 2009.
[42] W.-L. Chao, S. Changpinyo, B. Gong, and F. Sha, "An empirical study and analysis of generalized zero-shot learning for object recognition in the wild," Eur. Conf. Comput. Vis., pp.52–68, 2016.

Masayuki Hiromoto received B.E. degree in Electrical and Electronic Engineering and M.Sc. and Ph.D. degrees in Communications and Computer Engineering from Kyoto University in 2006, 2007, and 2009 respectively. He was a JSPS research fellow from 2009 to 2010 in Kyoto University, worked with Panasonic Corp. from 2010 to 2013, and was an assistant professor from 2013 and a senior lecturer from 2017 in Kyoto University. In 2019, he joined Fujitsu Research, where he is currently a senior researcher. His research interests include VLSI design methodology, computer vision, and artificial intelligence. He is also a member of IEEE and IPSJ.

Hisanao Akima received the B.E., M.S., and Ph.D. degrees from Tohoku University, Sendai, Japan, in 2000, 2002, and 2006, respectively. From April 2006 to March 2013, he was with Central Research Laboratory, Hitachi, Ltd., Tokyo, Japan, where he worked on spherical aberration corrector for electron microscopes. From April 2013 to March 2018, he was an assistant professor in the Research Institute of Electrical Communication, Tohoku University. Currently, he is a senior researcher in Fujitsu Ltd., Tokyo, Japan. His current research interests include artificial intelligence and brain-inspired computing.

Teruo Ishihara received the B.E. degree in Electrical Communication from Tohoku University, Sendai Japan in 1987. He joined Fujitsu Ltd., Kawasaki Japan in 1987 and has been engaged in development of DSPs for communication systems. Currently, he is an expert in Fujitsu Ltd. and is engaged in research and development of new computer architectures.

Takuji Yamamoto received the B.S. degree from Keio University in 1986. In 1986, he joined Fujitsu Laboratories Ltd. Japan, where he is currently a Senior Expert. He has been engaged in research on high-speed communication systems and computing systems. His current interests include the next generation computing with high performance and low power.