Zero-Shot Learning via Threefold Semantic Mapping Paths

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Zero-shot learning aims to recognize newly appeared instances of unseen classes with only labeled instances from another set of seen classes. In this paper, we propose a novel zero-shot learning method based on threefold semantic mapping paths: attribute classification, semantic representation, and label word vectors. Our discriminative model compromises the merits of both attribute extraction and semantic properties. We conduct experiments on two standard image datasets, Unstructured Social Activity Attribute and Animals with Attributes, by comparing the proposed approach to the state-of-the-art zero-shot learning methods and the traditional supervised learning.

1 Introduction

Object Recognition is a fundamental task in computer vision field. It usually relies on powerful feature extraction mechanisms based on a large scale of images. For example, the traditional image classification frameworks, such as deep neural networks (DNN)[1] rely on a large number of training samples to build statistical models. However, with the rapid development of image collection ability, the class categories expand quickly as the increment of image collections. In many realistic applications, collecting images for ever-increasing new classes is unattainable.

To meet the cutting-edge needs required for building prediction systems over the new-coming classes, zero-shot learning (ZSL), which transfers information from labeled instances of seen classes to recognize the new classes that have not been seen in training data, has recently aroused people‘s special attention in the research community. The fundamental difficulty of ZSL is that training cannot be guided by the end goal of the classifier.

State-of-the-art methods rely on assisted information to realize the classification, such as object attributes [2]. While image annotation using such attributes can be performed by naive users, more recently, methods have emerged that automatically infer such an intermediate "semantic" representation. Domain experts have to compile the initial list of discriminative attributes for a fixed set of classes and have to revise this list whenever new classes are added. Among these, some have cast the problem as joint alignment of the data using graph structures [3, 4] or directly using regularized sparse representations [5, 6]. Several recent works therefore evaluated alternatives, such as distributed text representations extracted from online text corpora. Such representations can be extracted automatically and are therefore less costly[7, 8]. Nevertheless, Most automatic methods perform at levels insufficient to support practical applications[7].

In this work, we investigate the advantages and disadvantages of the two approaches for implementations based on deep learning and CNNs. We propose a method by training a visual recognition model with both labeled images and semantic information from unannotated text data. We present the images with both attributes and intermediate semantic representation. Hence, we get three semantic mapping paths from an labeled image: attributes, semantic representation and word vectors learned from a comparatively large and independent dataset.

The potential applications of our model is in object recognition, especially recognizing objects with physical labels, such as the commodities. We can first recognize the words on a picture, and then recognize the object of the picture with the information extracted from the word. On the other hand, we assume that the words in a picture would have some kinds of relationship with the object we are going to recognize. Hence, the aid of semantics information contributes to the recognition accuracy.

2 Related Work

The state-of-the-art of zero-shot learning can be classified into three categories: projection, attributes, and semantics.

2.1 Zero-Shot Learning

In zero-shot learning setting test and training class sets are disjoint [9] which can be tackled by solving related sub-problems, e.g. learning intermediate attribute classifiers [10] and learning a mixture of seen class proportions [3], or by a direct approach, such as compatibility learning frameworks.

2.2 Projection

Many ZSL methods explore semantic relations between seen classes and unseen classes to achieve the goal of automatically categorizing instances into unseen classes. The visual feature projection methods first train a projection model based on the training instances and the attribute vectors (or semantic embeddings) of the training classes. Then given a test instance, they project the instance onto the semantic space and assign it into one of the unseen classes by comparing the semantic output with the prototypes of unseen classes. Many different projection strategies have been adopted in the literature, including attribute direct prediction [10], linear mapping, convolutional neural networks, and simple two layer linear networks. These methods however fail to take the unlabeled instances from the unseen classes into account during the projection function learning process.[11]

2.3 Attributes

Besides manually specified attributes [10], several researchers have explored various attribute applications and attempted to automatically discover these attributes [12, 13]. Recent approaches model attributes in a continuous space [14]. The main idea of these approaches is to learn a transformation matrix W that correlates attributes to images. We name these methods transformation-based approaches.

Other zero-shot approaches used graph/hyper-graphs built on attributes and class labels [9]. In contrast to graph based approaches, transformation-based approaches have recently shown better performance and are meanwhile simpler and more efficient on fine-grained recognition [14].

2.4 Semantics

A variety of zero-shot learning models have been proposed recently. They use various semantic spaces. Attribute space is the most widely used. However, for largescale problems, annotating attributes for each class becomes difficult. Recently, semantic word vector space has started to gain popularity especially in large-scale zero-shot learning [15]. Better scalability is typically the motivation as no manually defined ontology is required and any class name can be represented as a word vector for free. Beyond semantic attribute or word vector, direct learning from textual descriptions of categories has also been attempted, e.g. Wikipedia articles, sentence descriptions. [16]

3 Methods

In this section, we mainly introduce both several state-of-the-art zero-shot learning methods and our proposed model.

3.1 Zero-Shot Learning

Definition Let $\mathbf{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_s\}$ and $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_u\}$ denote a set of s seen and u unseen class labels, and they are disjoint $\mathbf{Y} \cap \mathbf{Z} = \emptyset$. Similarly $\mathbf{S_Y} = \{\mathbf{s}_1, ..., \mathbf{s}_s\} \in \mathbb{R}^{s \times k}$ and $\mathbf{S_Z} = \{\mathbf{s}_1, ..., \mathbf{s}_u\} \in \mathbb{R}^{u \times k}$ denote the corresponding seen and unseen class semantic representations (e.g. k-dimensional attribute vector). Given training data with N number of samples $\mathbf{X_Y} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{s}_i)\} \in \mathbb{R}^{d \times k}$, where $\mathbf{x_i}$ is a d-dimensional visual feature vector extracted from the i-th training image from one of the seen classes, zero-shot learning aims to learn a classifier $\mathbf{X_Z} \to \mathbf{Z}$ to predict the label of the image coming from unseen classes, where $\mathbf{X_Z} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{s}_i)\}$ is the test data and $\mathbf{z}_i$ and $\mathbf{s}_i$ are unknown.[16]

3.2 Attribute-Based Classification

Definition Let $(x_1, l_1), ..., (x_n, l_n) \subset \mathcal{X} \times \mathcal{Y}$ be training samples where $\mathcal{X}$ is an arbitrary feature space and $\mathcal{Y} = y_1, ..., y_K$ consists of K discrete classes. The task is to learn a classifier $f : \mathcal{X} \to \mathcal{Z}$ for a label set $\mathcal{Z} = z_1, ..., z_L$ that is disjoint from $\mathcal{Y}^1$. If for each class $z \in \mathcal{Z}$ and $y \in \mathcal{Y}$ an attribute representation $a \in \mathcal{A}$ is available, then we can learn a non-trivial classifier $\alpha : \mathcal{X} \to \mathcal{Z}$ by transferring information between $\mathcal{Y}$ and $\mathcal{Z}$ through $\mathcal{A}$.[10]

There are two generic methods to integrate attributes into multi-class classification: *direct attribute prediction* and *indirect attribute prediction*. Also, we introduce a method, *semantically consistent regularization*, which is based on direct attribute prediction.

3.2.1 Direct attribute prediction (DAP)

Direct attribute prediction (DAP)[10] uses an in between layer of attribute variables to decouple the images from the layer of labels. During training, the output class label of each sample induces a deterministic labeling of the attribute layer. Consequently, any supervised learning method can be used to learn perattribute parameters β_m. At test time, these allow the prediction of attribute values for each test sample, from which the test class label are inferred. Note that the classes during testing can differ from the classes used for training, as long as the coupling attribute layer is determined in a way that does not require a training phase.[10]

3.2.2 Indirect attribute prediction (IAP)

Indirect attribute prediction (IAP) uses the attributes to transfer knowledge between classes, but the attributes form a connecting layer between two layers of labels, one for classes that are known at training time and one for classes that are not. The training phase of IAP is ordinary multi-class classification. At test time, the predictions for all training classes induce a labeling of the attribute layer, from which a labeling over the test classes can be inferred.[10]

3.2.3 Semantically Consistent Regularization (SCoRe)

Given a training set of images $\mathbf{x}^{(i)}$, attribute labels $(\mathbf{s}_1^{(i)}, ..., \mathbf{s}_Q^{(i)})$, and class labels $y^{(i)}$, the regularizers of the previous sections are combined into the SCoRe objective

$$\begin{aligned} \text{minimize}_{\Theta, \mathbf{T}, \mathbf{W}} & \sum_i L(h(\mathbf{x}^{(i)}; \Theta, \mathbf{T}, \mathbf{W}), y^{(i)}) \\ & + \lambda \sum_i \sum_k L_b(f_k(\mathbf{x}^{(i)}; \mathbf{t}_k, \Theta), \mathbf{s}_k^{(i)}) \\ & + \beta \Omega[\mathbf{W}] \end{aligned}$$

where $h(\cdot)$ is

$$h(\mathbf{x}; \Theta, \mathbf{T}, \mathbf{W}) = \mathbf{W}^T f(\mathbf{x}) = \mathbf{W}^T T^T \theta(\mathbf{x}; \Theta),$$

$f_k(\mathbf{x}^{(i)}; \mathbf{t}_k, \Theta) = \mathbf{t}_k^T \theta(\mathbf{x}; \Theta)$ is the k^{th} semantic predictor, $\Omega[\mathbf{W}]$ the codeword regularizer

$$\Omega[\mathbf{W}] = \frac{1}{2} \sum_{c=1}^{C} ||\mathbf{w}_c - \phi(c)||^2$$

, and λ and β Lagrange multipliers that control the tightness of the regularization constraints.

Depending on the value of these multipliers, SCoRe can learn a standard CNN, Deep-RIS[17], or Deep-RULE[17]. When $\lambda = \beta = 0$, all the regularization constraints are disregarded and the classifier is a standard recognizer for the training classes. Increasing λ and β improves its transfer ability. On one hand, regardless of β, increasing λ makes SCoRe more like Deep-RIS. For large values of β, the learning algorithm emphasizes the similarity between classification and semantic codes, trading off classification performance for semantic alignment. Finally, when both λ and β are nonzero, SCoRe learns the classifier that best satisfies the corresponding trade-off between the three goals: recognition, attribute predictions, and alignment with the semantic code. [17]

3.3 Semantics

3.3.1 Deep Visual-Semantic Embedding Model (DeViSE)

Our deep visual-semantic embedding model (DeViSE) is initialized from these two pre-trained neural network models. The embedding vectors learned by the language model are unit normed and used to map label terms into target vector representations.

The core visual model, with its softmax prediction layer now removed, is trained to predict these vectors for each image, by means of a projection layer and a similarity metric. The projection layer is a linear transformation that maps the 4,096-D representation at the top of our core visual model into the 500- or 1,000-D representation native to our language model.[15]

3.4 Convex Combination of Semantic Embedding (ConSE)

ConSE follows the classic machine learning approach, and learns a classifier from training inputs to training labels instead of explicitly learning a regression function.[18]

3.5 Word Embedding

Word-embedding plays an essential part in natural language processing. It encodes the semantic meanings of a word into a real-valued low-dimensional vector. Recent years have witnessed major advances of word embedding.

3.5.1 Global Vectors for Word Representation

As anther embedding algorithm that shares worldwide popularity as does word2vec, Global Vectors for Word Representation (GloVe) stands out as a matrix factorization algorithm and features its computational convince.[1].

The key idea behind GloVe is to harness the statistics of word occurrences in a corpus, which is the primary source of information available to all unsupervised methods for learning word representations.[19]

Denote X_{ij} as the number of times word j occurs in the context of word i. Let $X_i = \sum_k X_{ik}$ be the number of times any word appears in the context of word i. Finally, let $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$ be the probability that word j appear in the context of i. Usually we scan our corpus in the following manner: for each term we look for context terms within some area defined by a $window\_size$ before the term and a $window\_size$ after the term. Define soft constraints for each word pair as

$$w_i^T w_j + b_i + b_j - \log X_{ij},$$

where w_i is the vector for the main word and w_j is that of context word. b_j and b_i are bias terms that are specific to each focus word and each context word, respectively. The loss function is defined as

$$J = \sum_{i,j} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2.$$

[1] https://nlp.stanford.edu/projects/glove/

Here f is a weighting function which help us to prevent learning only from extremely common word pairs. A common choice is

$$f(X_{ij}) = (\frac{X_{ij}}{X_{max}})^{\alpha}\mathbf{I}(X_{ij} < X_{max}) + 1\mathbf{I}(X_{ij} \geq X_{max}).$$

3.6 Threefold Semantic Mapping Paths

We propose a new framework, namely threefold semantic mapping paths (TSMP), to solve this joint optimization problem by compromising the three methods mentioned above 1.

The whole architecture of our model. From bottom to top, we first use CNNs to select some features of an image. Then we use the features in two ways. One is to extract attributes via full connected neural network. The other is to extract its visual semantic mapping path. Meanwhile, we get the word vector from the image label by using a language model trained with large scale text. We combine these three semantic mapping path (attributes, visual-semantic vectors, and word vector) to a feature vector. Finally, we train kNN part with the target.

3.6.1 Implement

The critical python codes of our Threefold Semantic Mapping Paths model is shown in appendix part.

4 Datasets and Evaluation Protocol

4.1 Datasets

In this paper, we do our experiments on two datasets for zero-shot learning tasks. One dataset is unstructured social activity attribute (USAA)[2] dataset, including both visual and audio attributes, which is for understanding event or activity happened in the video.[20] The other dataset is a large scale dataset, Animals with Attributes (AwA)[3], of over 30,000 animal images that match the 50 classes in Osherson's classic table of how strongly humans associate 85 semantic attributes with animal classes.[10] It is a coarse-grained dataset that is medium-scale in terms of the number of images, i.e. 30, 475 and small-scale in terms of number of classes, i.e. 50. AwA has 85 attributes.[21]23

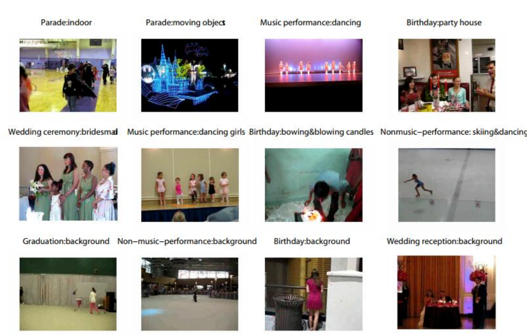

Figure 2: USAA

[2] http://yanweifu.github.io/USAA/download/

[3] https://cvml.ist.ac.at/AwA2/

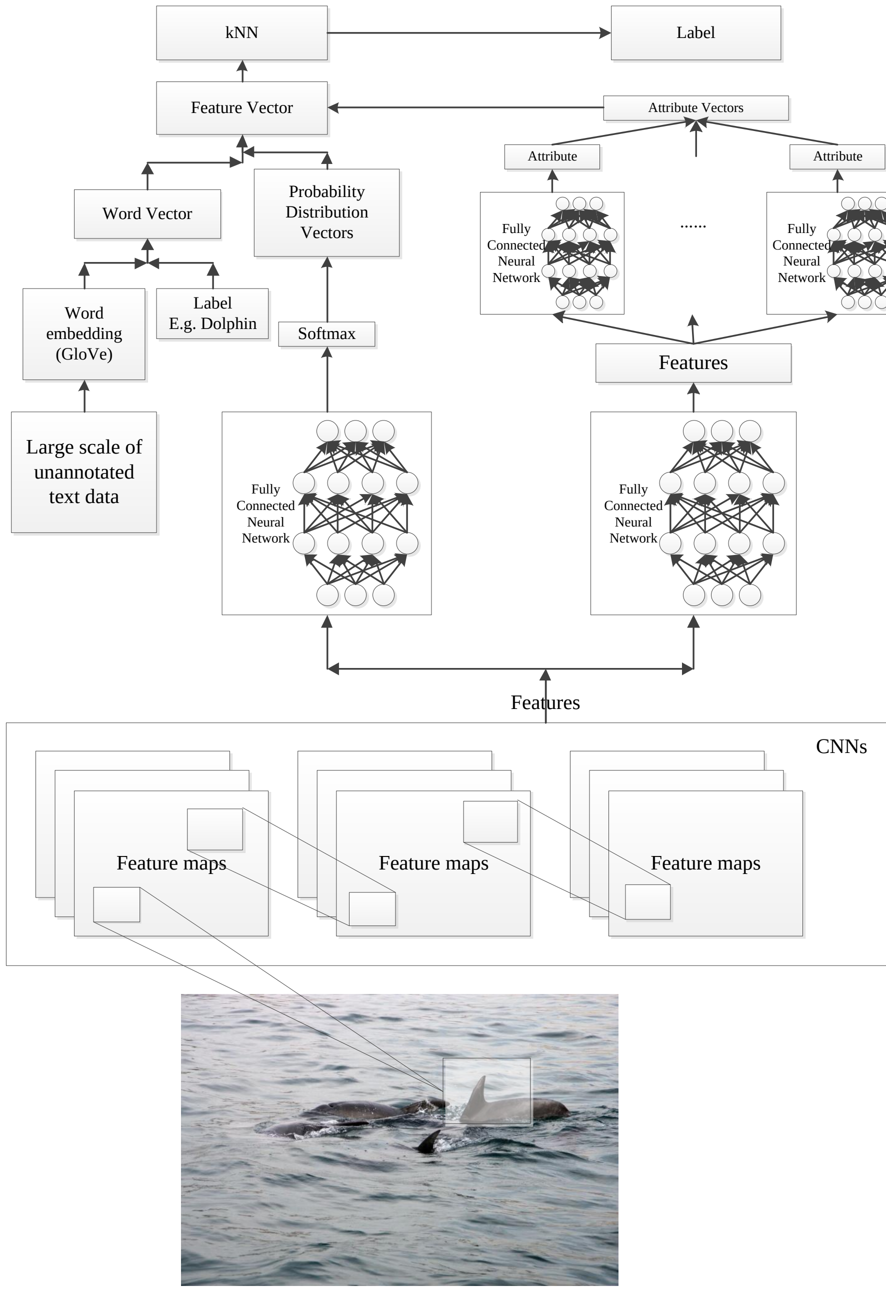

Figure 1: The whole architecture of our model. From bottom to top, we first use CNNs to select some features of an image. Then we use the features in two ways. One is to extract attributes via full connected neural network. The other is to extract its visual semantic mapping path. Meanwhile, we get the word vector from the image label by using a language model trained with large scale text. We combine these three semantic mapping path (attributes, visual-semantic vectors, and word vector) to a feature vector. Finally, we train kNN part with the target.

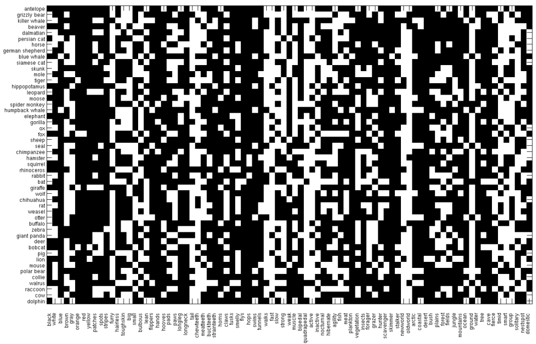

Figure 3: AwA: Label-Attribute

4.2 Evaluation Protocol

On USAA dataset we compare both supervised learning and zero-shot learning scenarios.

On AwA dataset, we use the unified protocol several components of generalized zero-shot learning evaluation protocols. According to [21], when measure the accuracy while training and validating, one of the common ways to follow is simply add the right-classified instances tighter. But when there are much more classes and the number of instances in each class is different, this way tends to evaluate the accuracy of the class which has the most instances, rather than the accuracy of the whole instance space. So the accuracy of each class should be rescaled, by the number of instances in the class and then be average together.

When comes to zero shot learning, the search space is no only the space of test space, but also training space. So when computing the accuracy, both training and test accuracy should be considered. That's why in the paper, the author proposed the harmonic mean to be the evaluation criteria, to get high accuracy on both training and test classes.

5 Experiments

5.1 Data Preparations

5.1.1 USAA Data

Since the USAA dataset's feature has been featured, and it is in a good structure. We don't need too much data preparation or clean on this case. Besides, some videos' values are all zero. We believe this may not be a real video. So we strip such values. And the size of this dataset is not so large, so we don't need to extract any subset.

5.1.2 AwA Data

This dataset is quite large(more than 13G). Considering that we don't have a very good machine to compute or train, if we use the whole dataset, it will compute quite slowly. So we extract only half of the training animal labels(40 classes) and half of the validation animal labels(5 classes). And for each animal, we just use 80% of all the related pictures to train our model randomly.

We will use the RGB model to read the images. Besides, considering that not all the pictures have the same size, so we resize them all to [224 * 224].

5.2 Experiments on USAA

We conduct experiments with both supervised learning and zero-shot learning on USAA.

5.2.1 Supervised Learning

We test linear regression, classification, KNN, NN, logistic regression, linear/RBF kernel SVM, neural network as well as tree-based methods on USAA. Table 1 shows the results.

Methods	Accuracy
Linear Regression	18.8
Naive Bayes	19.7
kNN	22.4
NN	21.9
Logistic Regression	23.2
SVM	25.8
Decision Tree	23.7
Neural Network	30.2
CNN	32.3

Table 1: Supervised Learning Methods on USAA

5.2.2 Zero-Shot Learning

We use three splits for ZSL, and the zero-shot (testing classes) are [1,2,4,7], [1,6,7,8],[2,4,5,6]. Since the dataset is a feature selected dataset, we generate zero-shot learning method without semantics, DAP, IAP and SCoRe on the dataset. Table 2 shows the results.

Methods	Accuracy
DAP	31.2
IAP	20.8
SCoRe	33.9
IAP with regularization	22.5

Table 2: Zero-Shot Learning Methods on USAA

5.3 Experiments on AwA

5.3.1 Supervised Learning

First, we use a pre-trained VGG19 to extract features. Then we test linear regression, classification, KNN, NN, logistic regression, linear/RBF kernel SVM, neural network as well as tree-based methods on AwA. Table 3 shows the results.

Methods	Accuracy
ResNet50	50.3
VGG16	59.2
VGG19	64.7

Table 3: Supervised Learning Methods on AwA

5.3.2 Zero-Shot Learning

We generate zero-shot learning method, DAP, IAP, SCoRe, IAP with regularization, DeViSE, ConSE on the dataset. Table 4 shows the results.

Methods	word2vec(bi)	without word2vec(bi)	word2vec(conti)	without word2vec(conti)
DAP	44.2	37.6	32.3	28.2
IAP	40.8	36.2	30.2	24.2
SCoRe	52.2	43.2	38.2	36.2
IAP with regularization	46.5	41.2	34.2	37.2

Table 4: Zero-Shot Learning Methods on AwA

5.3.3 Threefold Semantic Mapping Paths

We test our Threefold Semantic Mapping Paths model on AwA and get accuracy of **54.7%**, which is higher than all the previous methods we tried before.

5.4 Analysis

5.4.1 Fine tuning

It is a general method to apply successful model to some new project, especially the image classification project. We can use models like VGG, ResNet to extract the space feature vector from images, then train the part behind that and fine tuning the top layers of the CNN models. It would save time and effort. We select VGG19 out of the successful models like InceptronV3, ResNet50, VGG16 and so on because we have to consider both the size and speed of the network as well as its performance. For our equipment, networks like InceptronV3 and ResNet50 are too expensive to implement. So finally we decided on VGG19, for its balance of performance and network size.

5.4.2 The influence of resizing these images

Since we don't have the same size of these pictures, so we resize them to the same. The thing is, after we resize, some attributes of a certain animal might change. Just like the attribute "patch" and "dot", if you narrow the "patches, it might look like "dot". So we guess the resize step might the inclunce the accuracy of our model.

5.4.3 Word2vec

In our model, we use the GloVe, which is better than the naive word2vec usually. The word2vec might introduce more sematic information of the class. Take the example of "frog" and "toad"(although there are not in the AwA dataset), we can find that their cosine similarity of their word embedding vectors is quite small. And what's interesting is that from the view of the whole natural world, they are very likely to each other.

5.4.4 Semantic Information and Attributes Formats

From the results above, we can easily find that the performance improves when adding word vectors to the model. That means semantic information plays an important part in the object recognition task. This makes sense since the more information we get, the more accurate our model would be.

Also, the discrete attributes performs better than the continuous ones. That might because of the high freedom degree of continuous model. Due to the limited sample set of the training dataset, continuous model might cause over-fitting.

5.4.5 Machine Restriction

According to the machine restriction, we can only train the model with some subset of dataset. Hence, the performance might be poorer than state-of-the-art. However, if we could run our model on GPU, the performance might be better.

6 Conclusions

In this paper, we proposed a novel zero-shot learning approach, Threefold Semantic Mapping Paths, to make full use the information of of attribute classification, semantic representation, and label word vectors. It performs better than all other previous methods we tried before.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *International Conference on Neural Information Processing Systems*, pp. 1097–1105, 2012.

[2] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth, "Describing objects by their attributes," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 1778–1785, 2009.

[3] S. Changpinyo, W. L. Chao, B. Gong, and F. Sha, "Synthesized classifiers for zero-shot learning," in *Computer Vision and Pattern Recognition*, pp. 5327–5336, 2016.

[4] Y. Fu, T. M. Hospedales, T. Xiang, and S. Gong, "Transductive multi-view zero-shot learning," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 37, no. 11, p. 2332, 2015.

[5] E. Kodirov, T. Xiang, Z. Fu, and S. Gong, "Unsupervised domain adaptation for zero-shot learning," in *IEEE International Conference on Computer Vision*, pp. 2452–2460, 2015.

[6] R. Qiao, L. Liu, C. Shen, and A. V. D. Hengel, "Less is more: zero-shot learning from online textual documents with noise suppression," 2016.

[7] S. Deutsch, S. Kolouri, K. Kim, Y. Owechko, and S. Soatto, "Zero shot learning via multi-scale manifold regularization," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[8] N. Karessli, Z. Akata, B. Schiele, and A. Bulling, "Gaze embeddings for zero-shot image classification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[9] S. Huang, M. Elhoseiny, A. Elgammal, and D. Yang, "Learning hypergraph-regularized attribute predictors," pp. 409–417, 2015.

[10] C. Lampert, H. Nickisch, and S. Harmeling, "Learning to detect unseen object classes by between-class attribute transfer," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[11] M. Ye and Y. Guo, "Zero-shot classification with discriminative semantic representation learning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[12] T. L. Berg, A. C. Berg, and J. Shih, "Automatic attribute discovery and characterization from noisy web data," in *Computer Vision - ECCV 2010, European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings*, pp. 663–676, 2010.

[13] T. Mensink, E. Gavves, and C. G. M. Snoek, "Costa: Co-occurrence statistics for zero-shot classification," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2441–2448, 2014.

[14] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid, "Label-embedding for attribute-based classification," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 819–826, 2013.

[15] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. Ranzato, and T. Mikolov, "Devise: a deep visual-semantic embedding model," in *International Conference on Neural Information Processing Systems*, pp. 2121–2129, 2013.

[16] E. Kodirov, T. Xiang, and S. Gong, "Semantic autoencoder for zero-shot learning," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[17] P. Morgado and N. Vasconcelos, "Semantically consistent regularization for zero-shot recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2037–2046, 2017.

[18] 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," *Eprint Arxiv*, 2013.

[19] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[20] Y. Fu, T. M. Hospedales, T. Xiang, and S. Gong, "Attribute learning for understanding unstructured social activity," in *Computer Vision – ECCV 2012* (A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, eds.), (Berlin, Heidelberg), pp. 530–543, Springer Berlin Heidelberg, 2012.

[21] Y. Xian, B. Schiele, and Z. Akata, "Zero-shot learning - the good, the bad and the ugly," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

A Critical Codes

A.1 Threefold Semantic Mapping Paths

```
class IndirectWordVecDirectAttribute_VGG19(object):
    def __init__(self, learning_rate, attribute_length,
                                          num_classes_all,
                                          num_classes_train,
                                          attri_list_all, WV_list_all,
                 attri_list_train, WV_list_train,
                 vgg19_npy_path=None,
                 trainable=False,
                 img_shape=224,
                 regul_DA=0.0,
                 regul_IW=0.0):
        """
        :param learning_rate: Learning rate, float.
        :param attribute_length: The length of attribute, int.
        :param num_classes_all: The number of classes in training and
                                                   validation set, int.
        :param num_classes_train: The number of classes in training
                                                   set, int.
        :param attri_list_all: The relationship of labels and
                                                   attributes in training and
                                                   validation set, the list of
                                                    attributes (np.array),
        shape = [num_classes_all, attribute_length].
        Caution: the attributes must be in the order according to the
                                                   labels.
        :param WV_list_all: The relationship of labels and word
                                                   vectors in training and
                                                   validation set, the list of
                                                    attributes (np.array),
        shape = [num_classes_all, word_vector_length].
        Caution: the attributes must be in the order according to the
                                                   labels.
        :param attri_list_train: The relationship of labels and
                                                   attributes in training set,
                                                    the list of attributes (np
                                                   .array),
        shape = [num_classes_train, attribute_length].
        Caution: the attributes must be in the order according to the
                                                   labels.
        :param WV_list_train: The relationship of labels and word
                                                   vectors in training set,
                                                   the list of attributes (np.
                                                   array),
        shape = [num_classes_train, word_vector_length].
        Caution: the attributes must be in the order according to the
                                                   labels.
        :param vgg19_npy_path: The path of "vgg19.npy".
        :param trainable: A bool tensor, whether the VGG is trainable.
        :param img_shape: The width or height of image, int.
        :param regul_DA: The rate of direct attribute learning
                                                   regularization, positive
                                                   float.
        :param regul_IW: The rate of indirect work vector learning
                                                   regularization, positive
                                                   float.
        """
        relu = tf.nn.relu
        tanh = tf.nn.tanh
        sigmoid = tf.nn.sigmoid
        BatchNormalization = tf.layers.batch_normalization
```

```
        dropout = tf.nn.dropout
        dense = tf.layers.dense
        softmax = tf.nn.softmax

        if vgg19_npy_path is not None:
            self.data_dict = np.load(vgg19_npy_path, encoding='latin1'
                                        ).item()
        else:
            self.data_dict = None

        self.var_dict = {}
        self.trainable = trainable
        self.learning_rate = learning_rate

        self.img_width = img_shape
        self.img_height = img_shape
        self.attribute_length = attribute_length
        self.num_classes_all = num_classes_all
        self.num_classes_train = num_classes_train
        self.attri_list_all = attri_list_all
        self.attri_list_train = attri_list_train
        self.WV_list_train = WV_list_train
        self.WV_list_all = WV_list_all

        self.img_tensor = tf.placeholder(tf.float32,
            shape=[None, self.img_width, self.img_height, 3])
        self.img_attribute = tf.placeholder(tf.float32,
            shape=[None, self.attribute_length])
        self.img_label_all = tf.placeholder(tf.float32,
            shape=[None, self.num_classes_all])
        self.img_label_train = tf.placeholder(tf.float32,
            shape=[None, self.num_classes_train])
        self.dropout = tf.placeholder(tf.float32)
        self.regul_DA = max(0., regul_DA)
        self.regul_IW = max(0., regul_IW)

        self.build(input_image=self.img_tensor, include_top=False)

        self.feature = tf.reshape(self.pool5, [-1, 25088])

        h_fc1 = dropout(sigmoid(BatchNormalization(
            dense(self.feature, 1024), axis=1, training=True)),
        keep_prob=1-self.dropout)

        # Direct Attribute Learning.
        self.predic_attr_ = dense(h_fc1, self.attribute_length)

        self.predic_attr = sigmoid(self.predic_attr_)

        # Indirect WordVec learning.
        self.predic_label_train_ = dense(
            self.feature, self.num_classes_train)

        self.predic_label_train = softmax(self.predic_label_train_)

        self.predic_WV = self.Get_WV(
            self.predic_label_train, self.WV_list_train)

        # predict label.
        self.predic_label = self.Get_label(
            self.predic_attr, self.predic_WV, self.attri_list_all,
                                        self.WV_list_all)

        self.acc = self.acc_label(self.img_label_all, self.
                                    predic_label)
```

```

        self.loss = (1.0 / (1.0 + self.regul_DA + self.regul_IW)) * tf
                                            .reduce_mean(tf.nn.
                                            sigmoid_cross_entropy_with_logits
                                            (labels=self.img_attribute,
                                             logits=self.predic_attr_))
                                             + (self.regul_DA / (1.0 +
                                            self.regul_DA + self.
                                            regul_IW)) * tf.reduce_mean
                                            ((self.img_label_all - self
                                            .predic_label) ** 2) + (
                                            self.regul_IW / (1.0 + self
                                            .regul_DA + self.regul_IW))
                                             * tf.reduce_mean(tf.nn.
                                            softmax_cross_entropy_with_logits
                                            (labels=self.
                                            img_label_train, logits=
                                            self.predic_label_train_))

        self.train_op = tf.train.AdamOptimizer(learning_rate=self.
                                            learning_rate).minimize(
                                            self.loss)

    def build(self, input_image, include_top=False,  train_mode=None):
        """
        Load variable from .npy file to build the VGG19.
        :param input_image: RGB image tensor: [batch, height, width, 3
                                            ]. Values scaled [0, 1].
        :param include_top: A bool tensor, whether to include the
                                            fully connected layers.
        :param train_mode: A bool tensor, usually a placeholder: if
                                            True, dropout will be
                                            turned on
        """

        VGG_MEAN = [103.939, 116.779, 123.68]

        input_image_scaled = input_image * 255.0

        # Convert RGB to BGR.
        red, green, blue = tf.split(axis=3, num_or_size_splits=3,
                                            value=input_image_scaled)

        assert red.get_shape().as_list()[1:] == [self.img_width, self.
                                            img_height, 1]
        assert green.get_shape().as_list()[1:] == [self.img_width,
                                            self.img_height, 1]
        assert blue.get_shape().as_list()[1:] == [self.img_width, self
                                            .img_height, 1]

        bgr_image = tf.concat(axis=3, values=[
            blue - VGG_MEAN[0],
            green - VGG_MEAN[1],
            red - VGG_MEAN[2],
        ])
        assert bgr_image.get_shape().as_list()[1:] == [224, 224, 3]

        self.conv1_1 = self.conv_layer(bgr_image, 3, 64, "conv1_1")
        self.conv1_2 = self.conv_layer(self.conv1_1, 64, 64, "conv1_2"
                                            )
        self.pool1 = self.max_pool(self.conv1_2, 'pool1')

        self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
        self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128, "
                                            conv2_2")
```

```
        self.pool2 = self.max_pool(self.conv2_2, 'pool2')

        self.conv3_1 = self.conv_layer(self.pool2, 128, 256, "conv3_1"
                                       )
        self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256, "
                                       conv3_2")
        self.conv3_3 = self.conv_layer(self.conv3_2, 256, 256, "
                                       conv3_3")
        self.conv3_4 = self.conv_layer(self.conv3_3, 256, 256, "
                                       conv3_4")
        self.pool3 = self.max_pool(self.conv3_4, 'pool3')

        self.conv4_1 = self.conv_layer(self.pool3, 256, 512, "conv4_1"
                                       )
        self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512, "
                                       conv4_2")
        self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512, "
                                       conv4_3")
        self.conv4_4 = self.conv_layer(self.conv4_3, 512, 512, "
                                       conv4_4")
        self.pool4 = self.max_pool(self.conv4_4, 'pool4')

        self.conv5_1 = self.conv_layer(self.pool4, 512, 512, "conv5_1"
                                       )
        self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512, "
                                       conv5_2")
        self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512, "
                                       conv5_3")
        self.conv5_4 = self.conv_layer(self.conv5_3, 512, 512, "
                                       conv5_4")
        self.pool5 = self.max_pool(self.conv5_4, 'pool5')

        if include_top:
            self.fc6 = self.fc_layer(self.pool5, 25088, 4096, "fc6")
                                               # 25088 = ((224 // (2
                                               ** 5)) ** 2) * 512
            self.relu6 = tf.nn.relu(self.fc6)
            if train_mode is not None:
                self.relu6 = tf.cond(train_mode, lambda: tf.nn.dropout
                                                   (self.relu6, self.
                                                   dropout), lambda:
                                                   self.relu6)
            elif self.trainable:
                self.relu6 = tf.nn.dropout(self.relu6, self.dropout)

            self.fc7 = self.fc_layer(self.relu6, 4096, 4096, "fc7")
            self.relu7 = tf.nn.relu(self.fc7)
            if train_mode is not None:
                self.relu7 = tf.cond(train_mode, lambda: tf.nn.dropout
                                                   (self.relu7, self.
                                                   dropout), lambda:
                                                   self.relu7)
            elif self.trainable:
                self.relu7 = tf.nn.dropout(self.relu7, self.dropout)

            self.fc8 = self.fc_layer(self.relu7, 4096, 1000, "fc8")

            self.prob = tf.nn.softmax(self.fc8, name="prob")

        self.data_dict = None

        return None
```

A.2 DAP

```
class DirectAttribute_VGG19(object):
    def __init__(self, learning_rate, attribute_length,
                                     num_classes_all, attri_list_all
                                     ,
                 vgg19_npy_path=None,
                 trainable=False,
                 img_shape=224,
                 regul=0.0):
        """
        :param learning_rate: Learning rate, float.
        :param attribute_length: The length of attribute, int.
        :param num_classes_all: The number of classes in training and
                                           validation set, int.
        :param attri_list_all: The relationship of labels and
                                           attributes, the list of
                                           attributes (np.array),
        shape = [num_classes, attribute_length].
        Caution: the attributes must be in the order according to the
                                           labels.
        :param vgg19_npy_path: The path of "vgg19.npy".
        :param trainable: A bool tensor, whether the VGG is trainable.
        :param img_shape: The width or height of image, int.
        :param regul: The rate of regularization, positive float.
        """
        relu = tf.nn.relu
        tanh = tf.nn.tanh
        sigmoid = tf.nn.sigmoid
        BatchNormalization = tf.layers.batch_normalization
        dropout = tf.nn.dropout
        dense = tf.layers.dense

        if vgg19_npy_path is not None:
            self.data_dict = np.load(vgg19_npy_path, encoding='latin1'
                                            ).item()
        else:
            self.data_dict = None

        self.var_dict = {}
        self.trainable = trainable
        self.learning_rate = learning_rate

        self.img_width = img_shape
        self.img_height = img_shape
        self.attribute_length = attribute_length
        self.num_classes_all = num_classes_all
        self.attri_list_all = attri_list_all

        self.img_tensor = tf.placeholder(tf.float32, shape=[None, self
                                           .img_width, self.img_height
                                           , 3])
        self.img_attribute = tf.placeholder(tf.float32, shape=[None,
                                           self.attribute_length])
        self.img_label_all = tf.placeholder(tf.float32, shape=[None,
                                           self.num_classes_all])
        self.dropout = tf.placeholder(tf.float32)
        self.regul = max(0., regul)

        self.build(input_image=self.img_tensor, include_top=False)

        self.feature = tf.reshape(self.pool5, [-1, 25088])

        h_fc1 = dropout(sigmoid(BatchNormalization(dense(self.feature,
                                            1024), axis=1, training=
```

```
                                              True)), keep_prob=1 - self.
                                              dropout)

        self.predic_attr_ = dense(h_fc1, self.attribute_length)

        self.predic_attr = sigmoid(self.predic_attr_)


        self.predic_label = self.Get_label(self.predic_attr, self.
                                              attri_list_all)

        self.acc = self.acc_label(self.img_label_all, self.
                                              predic_label)

        self.loss = (1.0/(1.0+self.regul)) * tf.reduce_mean(tf.nn.
                                              sigmoid_cross_entropy_with_logits
                                              (labels=self.img_attribute,
                                               logits=self.predic_attr_))
                                               + (self.regul / (1.0 +
                                              self.regul)) * tf.
                                              reduce_mean((self.
                                              img_label_all-self.
                                              predic_label)**2)

        self.train_op = tf.train.AdamOptimizer(learning_rate=self.
                                              learning_rate).minimize(
                                              self.loss)

    def build(self, input_image, include_top=False,  train_mode=None):
        """
        Load variable from .npy file to build the VGG19.
        :param input_image: RGB image tensor: [batch, height, width, 3
                                              ]. Values scaled [0, 1].
        :param include_top: A bool tensor, whether to include the
                                              fully connected layers.
        :param train_mode: A bool tensor, usually a placeholder: if
                                              True, dropout will be
                                              turned on
        """

        VGG_MEAN = [103.939, 116.779, 123.68]

        input_image_scaled = input_image * 255.0

        red, green, blue = tf.split(axis=3, num_or_size_splits=3,
                                              value=input_image_scaled)

        assert red.get_shape().as_list()[1:] == [self.img_width, self.
                                              img_height, 1]
        assert green.get_shape().as_list()[1:] == [self.img_width,
                                              self.img_height, 1]
        assert blue.get_shape().as_list()[1:] == [self.img_width, self
                                              .img_height, 1]

        bgr_image = tf.concat(axis=3, values=[
            blue - VGG_MEAN[0],
            green - VGG_MEAN[1],
            red - VGG_MEAN[2],
        ])
        assert bgr_image.get_shape().as_list()[1:] == [224, 224, 3]

        self.conv1_1 = self.conv_layer(bgr_image, 3, 64, "conv1_1")
        self.conv1_2 = self.conv_layer(self.conv1_1, 64, 64, "conv1_2"
                                              )
        self.pool1 = self.max_pool(self.conv1_2, 'pool1')
```

```

        self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
        self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128, "conv2_2")
        self.pool2 = self.max_pool(self.conv2_2, 'pool2')

        self.conv3_1 = self.conv_layer(self.pool2, 128, 256, "conv3_1")
        self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256, "conv3_2")
        self.conv3_3 = self.conv_layer(self.conv3_2, 256, 256, "conv3_3")
        self.conv3_4 = self.conv_layer(self.conv3_3, 256, 256, "conv3_4")
        self.pool3 = self.max_pool(self.conv3_4, 'pool3')

        self.conv4_1 = self.conv_layer(self.pool3, 256, 512, "conv4_1")
        self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512, "conv4_2")
        self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512, "conv4_3")
        self.conv4_4 = self.conv_layer(self.conv4_3, 512, 512, "conv4_4")
        self.pool4 = self.max_pool(self.conv4_4, 'pool4')

        self.conv5_1 = self.conv_layer(self.pool4, 512, 512, "conv5_1")
        self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512, "conv5_2")
        self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512, "conv5_3")
        self.conv5_4 = self.conv_layer(self.conv5_3, 512, 512, "conv5_4")
        self.pool5 = self.max_pool(self.conv5_4, 'pool5')

        if include_top:
            self.fc6 = self.fc_layer(self.pool5, 25088, 4096, "fc6") # 25088 = ((224 // (2 ** 5)) ** 2) * 512
            self.relu6 = tf.nn.relu(self.fc6)
            if train_mode is not None:
                self.relu6 = tf.cond(train_mode, lambda: tf.nn.dropout(self.relu6, self.dropout), lambda: self.relu6)
            elif self.trainable:
                self.relu6 = tf.nn.dropout(self.relu6, self.dropout)

            self.fc7 = self.fc_layer(self.relu6, 4096, 4096, "fc7")
            self.relu7 = tf.nn.relu(self.fc7)
            if train_mode is not None:
                self.relu7 = tf.cond(train_mode, lambda: tf.nn.dropout(self.relu7, self.dropout), lambda: self.relu7)
            elif self.trainable:
                self.relu7 = tf.nn.dropout(self.relu7, self.dropout)

            self.fc8 = self.fc_layer(self.relu7, 4096, 1000, "fc8")

            self.prob = tf.nn.softmax(self.fc8, name="prob")

        self.data_dict = None

```

```
        return None
```

A.3 IAP

```
class IndirectAttribute_VGG19(object):
    def __init__(self, learning_rate, attribute_length,
                                        num_classes_all,
                                        num_classes_train,
                                        attri_list_all,
                                        attri_list_train,
                 vgg19_npy_path=None,
                 trainable=False,
                 img_shape=224,
                 regul=0.0):
        """
        :param learning_rate: Learning rate, float.
        :param attribute_length: The length of attribute, int.
        :param num_classes_all: The number of classes in training and
                                            validation set, int.
        :param num_classes_train: The number of classes in training
                                            set, int.
        :param attri_list_all: The relationship of labels and
                                            attributes in training and
                                            validation set, the list of
                                             attributes (np.array),
        shape = [num_classes_all, attribute_length].
        Caution: the attributes must be in the order according to the
                                            labels.
        :param attri_list_train: The relationship of labels and
                                            attributes in training set,
                                             the list of attributes (np
                                            .array),
        shape = [num_classes_train, attribute_length].
        Caution: the attributes must be in the order according to the
                                            labels.
        :param vgg19_npy_path: The path of "vgg19.npy".
        :param trainable: A bool tensor, whether the VGG is trainable.
        :param img_shape: The width or height of image, int.
        :param regul: The rate of regularization, positive float.
        """
        relu = tf.nn.relu
        tanh = tf.nn.tanh
        sigmoid = tf.nn.sigmoid
        BatchNormalization = tf.layers.batch_normalization
        dropout = tf.nn.dropout
        softmax = tf.nn.softmax
        dense = tf.layers.dense

        self.img_width = img_shape
        self.img_height = img_shape
        self.attribute_length = attribute_length
        self.num_classes_all = num_classes_all
        self.num_classes_train = num_classes_train
        self.attri_list_all = attri_list_all
        self.attri_list_train = attri_list_train

        if vgg19_npy_path is not None:
            self.data_dict = np.load(vgg19_npy_path, encoding='latin1'
                                                   ).item()
        else:
            self.data_dict = None

        self.var_dict = {}
```

```
        self.trainable = trainable
        self.learning_rate = learning_rate
        self.regul = max(0., regul)

        self.img_tensor = tf.placeholder(tf.float32, shape=[None, self
                                          .img_width, self.img_height
                                          , 3])
        self.img_attribute = tf.placeholder(tf.float32, shape=[None,
                                          self.attribute_length])
        self.img_label_all = tf.placeholder(tf.float32, shape=[None,
                                          self.num_classes_all])
        self.img_label_train = tf.placeholder(tf.float32, shape=[None,
                                           self.num_classes_train])
        self.dropout = tf.placeholder(tf.float32)

        self.build(input_image=self.img_tensor, include_top=False)

        self.feature = tf.reshape(self.pool5, [-1, 25088])

        h_fc1 = dropout(sigmoid(BatchNormalization(dense(self.feature,
                                           1024), axis=1, training=
                                          True)), keep_prob=1 - self.
                                          dropout)

        self.predic_label_train_ = dense(h_fc1, self.num_classes_train
                                          )

        self.predic_label_train = softmax(self.predic_label_train_)

        self.predic_attr = self.Get_attr(self.predic_label_train, self
                                          .attri_list_train)

        self.predic_label = self.Get_label(self.predic_attr, self.
                                          attri_list_all)

        self.acc = self.acc_label(self.img_label_all, self.
                                          predic_label)

        self.loss = (1.0/(1.0+self.regul)) * tf.reduce_mean(tf.nn.
                                          softmax_cross_entropy_with_logits
                                          (labels=self.
                                          img_label_train, logits=
                                          self.predic_label_train_))
                                          + (self.regul / (1.0 + self
                                          .regul)) * tf.reduce_mean(
                                          tf.nn.
                                          sigmoid_cross_entropy_with_logits
                                          (labels=self.img_attribute,
                                           logits=self.predic_attr))

        self.train_op = tf.train.AdamOptimizer(learning_rate=self.
                                          learning_rate).minimize(
                                          self.loss)

    def build(self, input_image, include_top=False,  train_mode=None):
        """
        Load variable from .npy file to build the VGG19.
        :param input_image: RGB image tensor: [batch, height, width, 3
                                          ]. Values scaled [0, 1].
        :param include_top: A bool tensor, whether to include the
                                          fully connected layers.
        :param train_mode: A bool tensor, usually a placeholder: if
                                          True, dropout will be
                                          turned on
        """
```

```
        VGG_MEAN = [103.939, 116.779, 123.68]

        input_image_scaled = input_image * 255.0

        red, green, blue = tf.split(axis=3, num_or_size_splits=3,
                                   value=input_image_scaled)

        assert red.get_shape().as_list()[1:] == [self.img_width, self.
                                   img_height, 1]
        assert green.get_shape().as_list()[1:] == [self.img_width,
                                   self.img_height, 1]
        assert blue.get_shape().as_list()[1:] == [self.img_width, self
                                   .img_height, 1]

        bgr_image = tf.concat(axis=3, values=[
            blue - VGG_MEAN[0],
            green - VGG_MEAN[1],
            red - VGG_MEAN[2],
        ])
        assert bgr_image.get_shape().as_list()[1:] == [224, 224, 3]

        self.conv1_1 = self.conv_layer(bgr_image, 3, 64, "conv1_1")
        self.conv1_2 = self.conv_layer(self.conv1_1, 64, 64, "conv1_2"
                                   )
        self.pool1 = self.max_pool(self.conv1_2, 'pool1')

        self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
        self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128, "
                                   conv2_2")
        self.pool2 = self.max_pool(self.conv2_2, 'pool2')

        self.conv3_1 = self.conv_layer(self.pool2, 128, 256, "conv3_1"
                                   )
        self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256, "
                                   conv3_2")
        self.conv3_3 = self.conv_layer(self.conv3_2, 256, 256, "
                                   conv3_3")
        self.conv3_4 = self.conv_layer(self.conv3_3, 256, 256, "
                                   conv3_4")
        self.pool3 = self.max_pool(self.conv3_4, 'pool3')

        self.conv4_1 = self.conv_layer(self.pool3, 256, 512, "conv4_1"
                                   )
        self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512, "
                                   conv4_2")
        self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512, "
                                   conv4_3")
        self.conv4_4 = self.conv_layer(self.conv4_3, 512, 512, "
                                   conv4_4")
        self.pool4 = self.max_pool(self.conv4_4, 'pool4')

        self.conv5_1 = self.conv_layer(self.pool4, 512, 512, "conv5_1"
                                   )
        self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512, "
                                   conv5_2")
        self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512, "
                                   conv5_3")
        self.conv5_4 = self.conv_layer(self.conv5_3, 512, 512, "
                                   conv5_4")
        self.pool5 = self.max_pool(self.conv5_4, 'pool5')

        if include_top:
            self.fc6 = self.fc_layer(self.pool5, 25088, 4096, "fc6")
            self.relu6 = tf.nn.relu(self.fc6)
```

```
            if train_mode is not None:
                self.relu6 = tf.cond(train_mode, lambda: tf.nn.dropout
                                                   (self.relu6, self.
                                                   dropout), lambda:
                                                   self.relu6)
            elif self.trainable:
                self.relu6 = tf.nn.dropout(self.relu6, self.dropout)

            self.fc7 = self.fc_layer(self.relu6, 4096, 4096, "fc7")
            self.relu7 = tf.nn.relu(self.fc7)
            if train_mode is not None:
                self.relu7 = tf.cond(train_mode, lambda: tf.nn.dropout
                                                   (self.relu7, self.
                                                   dropout), lambda:
                                                   self.relu7)
            elif self.trainable:
                self.relu7 = tf.nn.dropout(self.relu7, self.dropout)

            self.fc8 = self.fc_layer(self.relu7, 4096, self.
                                               num_classes_train, "fc8
                                               ")

            self.prob = tf.nn.softmax(self.fc8, name="prob")

        self.data_dict = None

        return None
```

A.4 Data Pre-processing

```
def Get_next_batch(Train_or_Vali, batch_size, epoch, use_word2vec=0,
                                   attribute = 'conti'):
  """
  Get next batch according to size.
  :param Train_or_Vali: Whether to train or validate the model.
  :param batch_size: The size of the training batch.
  :param use_word2vec: If use the word2vec feature as an attribute
  :return: training data, shape = [batch_size, img_width, img_height,
                                   img_path],
  attribute, shape = [batch_size, attribute_length]
  and label(one-hot), shape = [batch_size, num_classes]
  for next batch, type = np.array.
  """
  img_batch_tensor = np.zeros((batch_size, img_width, img_height, 3))
  if not use_word2vec:
    attribute_batch_tensor = np.ndarray((batch_size, 85))
  else:
    attribute_batch_tensor = np.ndarray((batch_size, 85 + use_word2vec
                                        ))
  # if not use_all_label:
  #   label_batch_tensor = np.ndarray((batch_size, 20))
  # else:
  label_batch_tensor = np.zeros((batch_size, 25))


  if Train_or_Vali == 'train':
    img_path = './Data/train_zsl/'
    for batch in range(batch_size):
      file_name = train_file_list[(batch_size*epoch + batch)%
                                  train_file_len]
      img_batch_tensor[batch] = image2tensor(img_path + file_name)
      label_name = file_name.split('_')[0]
      if attribute == 'bi':
```

```
        if not use_word2vec:
          attribute_batch_tensor[batch] = attribute_bi_dict[label_name
                                             ]
        else:
          attribute_batch_tensor[batch][:85] = attribute_bi_dict[
                                             label_name]
          attribute_batch_tensor[batch][85:] = glove(label_name,
                                             use_word2vec)
      elif attribute == 'conti':
        if not use_word2vec:
          attribute_batch_tensor[batch] = attribute_conti_dict[
                                             label_name]
        else:
          attribute_batch_tensor[batch][:85] = attribute_conti_dict[
                                             label_name]
          attribute_batch_tensor[batch][85:] = glove(label_name,
                                             use_word2vec)
      label_batch_tensor[batch][classes_new_dict[label_name]] = 1

  elif Train_or_Vali == 'validation':
    img_path = './Data/validation_zsl/'
    for batch in range(batch_size):
      file_name = vali_file_list[(batch_size*epoch + batch)%
                                         vali_file_len]
      img_batch_tensor[batch] = image2tensor(img_path + file_name)
      label_name = file_name.split('_')[0]
      if attribute == 'bi':
        if not use_word2vec:
          attribute_batch_tensor[batch] = attribute_bi_dict[label_name
                                             ]
        else:
          attribute_batch_tensor[batch][:85] = attribute_bi_dict[
                                             label_name]
          attribute_batch_tensor[batch][85:] = glove(label_name,
                                             use_word2vec)
      elif attribute == 'conti':
        if not use_word2vec:
          attribute_batch_tensor[batch] = attribute_conti_dict[
                                             label_name]
        else:
          attribute_batch_tensor[batch][:85] = attribute_conti_dict[
                                             label_name]
          attribute_batch_tensor[batch][85:] = glove(label_name,
                                             use_word2vec)
      label_batch_tensor[batch][classes_new_dict[label_name]] = 1
  else:
    print("ERROR!You should input train or validation")

  return img_batch_tensor.astype('float32'), attribute_batch_tensor.
                                     astype('float32'),
                                     label_batch_tensor.astype('
                                     float32')
```