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 22 attributes [2]. While image annotation using such attributes can be performed by naive users, 23 more recently, methods have emerged that automatically infer such an intermediate "semantic" 24 representation. Domain experts have to compile the initial list of discriminative attributes for a 25 fixed set of classes and have to revise this list whenever new classes are added. Among these, some 26 have cast the problem as joint alignment of the data using graph structures [3, 4] or directly using 27 28 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 29 extracted automatically and are therefore less costly [7, 8]. Nevertheless, Most automatic methods 30 perform at levels insufficient to support practical applications[7]. 31

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
- 37 from a comparatively large and independent dataset.
- 38 The potential applications of our model is in object recognition, especially recognizing objects with
- 39 physical labels, such as the commodities. We can first recognize the words on a picture, and then
- 40 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.

43 **2 Related Work**

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

46 2.1 Zero-Shot Learning

- 47 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.

50 2.2 Projection

- 51 Many ZSL methods explore semantic relations between seen classes and unseen classes to achieve
- 52 the goal of automatically categorizing instances into unseen classes. The visual feature projection
- 53 methods first train a projection model based on the training instances and the attribute vectors (or
- 54 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
- 57 literature, including attribute direct prediction [10], linear mapping, convolutional neural networks,
- 58 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]

60 2.3 Attributes

- Besides manually specified attributes [10], several researchers have explored various attribute appli-
- 62 cations and attempted to automatically discover these attributes [12, 13]. Recent approaches model
- 63 attributes in a continuous space [14]. The main idea of these approaches is to learn a transforma-
- 64 tion matrix W that correlates attributes to images. We name these methods transformation-based
- 65 approaches.
- 66 Other zero-shot approaches used graph/hyper-graphs built on attributes and class labels [9]. In
- 67 contrast to graph based approaches, transformation-based approaches have recently shown better
- 68 performance and are meanwhile simpler and more efficient on fine-grained recognition [14].

69 2.4 Semantics

- A variety of zero-shot learning models have been proposed recently. They use various semantic spaces.
- 71 Attribute space is the most widely used. However, for largescale problems, annotating attributes for
- 72 each class becomes difficult. Recently, semantic word vector space has started to gain popularity
- 73 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
- 75 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]

77 3 Methods

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

o 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 $\mathbf{S}_{\mathbf{Y}} = \{\mathbf{s}_1, ..., \mathbf{s}_s\} \in \mathbb{R}^{s \times k}$ and $\mathbf{S}_{\mathbf{Z}} = \{\mathbf{s}_1, ..., \mathbf{s}_u\} \in \mathbb{R}^{u \times k}$ denote the corresponding seen and unseen class semantic representations (e.g. k-dimensional attribute vector). Given training data with N number of samples $\mathbf{X}_{\mathbf{Y}} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{s}_i)\} \in \mathbb{R}^{d \times k}$, 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 $\mathbf{X}_{\mathbf{Z}} \to \mathbf{Z}$ to predict the label of the image coming from unseen classes, where $\mathbf{X}_{\mathbf{Z}} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{s}_i)\}$ is the test data and \mathbf{z}_i and \mathbf{s}_i are unknown.[16]

89 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.

98 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]

106 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]

112 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

minimize<sub>$$\Theta$$
, \mathbf{T} , \mathbf{W} $\sum_{i} L(h(\mathbf{x}^{(i)}; \Theta, \mathbf{T}, \mathbf{W}), y^{(i)})$
 $+ \lambda \sum_{i} \sum_{k} L_{b}(f_{k}(\mathbf{x}^{(i)}; \mathbf{t}_{k}, \Theta), \mathbf{s}_{k}^{(i)})$
 $+ \beta \Omega[\mathbf{W}]$</sub>

where $h(\cdot)$ is

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

116 $f_k(\mathbf{x}^{(i)}; \mathbf{t}_k, \Theta) = \mathbf{t}_k^T \theta(\mathbf{x}; \Theta)$ 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 118 Deep-RULE[17]. When $\lambda = \beta = 0$, all the regularization constraints are disregarded and the 119 classifier is a standard recognizer for the training classes. Increasing λ and β improves its transfer 120 ability. On one hand, regardless of β , increasing λ makes SCoRe more like Deep-RIS. For large 121 values of β , the learning algorithm emphasizes the similarity between classification and semantic 122 codes, trading off classification performance for semantic alignment. Finally, when both λ and β are 123 nonzero, SCoRe learns the classifier that best satisfies the corresponding trade-off between the three 124 goals: recognition, attribute predictions, and alignment with the semantic code. [17] 125

3.3 Semantics

126

127

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]

135 3.4 Convex Combination of Semantic Embedding (ConSE)

136 ConSE follows the classic machine learning approach, and learns a classifier from training inputs to 137 training labels instead of explicitly learning a regression function.[18]

138 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.

2 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

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

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

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

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
- 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.

167 **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

170 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

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

https://cvml.ist.ac.at/AwA2/

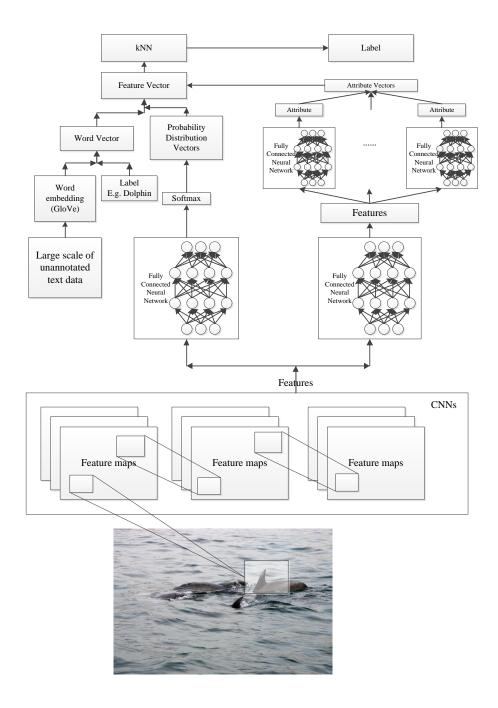


Figure 1: The whole architecture of our model. From bottom to top, we first use CNNs to select some features of an image. Then we use the features in two ways. One is to extract attributes via full connected neural network. The other is to extract its visual semantic mapping path. Meanwhile, we get the 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

178 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.

191 5 Experiments

5.1 Data Preparations

93 5.1.1 USAA Data

192

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.

198 **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].

205 5.2 Experiments on USAA

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

207 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

210 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

214 5.3 Experiments on AwA

215 5.3.1 Supervised Learning

- 216 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

219 5.3.2 Zero-Shot Learning

- 220 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

- 223 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 5.4 Analysis

226 **5.4.1** Fine tuning

- 227 It is a general method to apply successful model to some new project, especially the image classifica-
- 228 tion 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.

234 5.4.2 The influence of resizing these images

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

239 5.4.3 Word2vec

245

- 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
- 250 freedom degree of continuous model. Due to the limited sample set of the training dataset, continuous
- 251 model might cause over-fitting.

252 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,
- 255 the performance might be better.

256 6 Conclusions

- 257 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.

260 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.

- ²⁶⁹ [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.
- 292 [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. 2441–294 2448, 2014.
- 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.
- 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.
- 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.
- 306 [18] M. Norouzi, T. Mikolov, S. Bengio, Y. Singer, J. Shlens, A. Frome, G. S. Corrado, and J. Dean, 307 "Zero-shot learning by convex combination of semantic embeddings," *Eprint Arxiv*, 2013.
- 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.

316 A Critical Codes

317

A.1 Threefold Semantic Mapping Paths

```
318
    class IndirectWordVecDirectAttribute_VGG19(object):
319
        def __init__(self, learning_rate, attribute_length,
320
321
                                              num_classes_all,
322
                                              num_classes_train,
                                              attri_list_all, WV_list_all,
323
324
                      attri_list_train, WV_list_train,
325
                      vgg19_npy_path=None,
                      trainable=False,
326
                      img\_shape=224,
327
                      regul_DA=0.0,
328
                      regul_IW=0.0):
329
330
            :param learning_rate: Learning rate, float.
331
            :param attribute_length: The length of attribute, int.
332
333
            :param num_classes_all: The number of classes in training and
334
                                                   validation set, int.
335
            :param num_classes_train: The number of classes in training
                                                   set. int.
336
            :param attri_list_all: The relationship of labels and
337
                                                   attributes in training and
338
339
                                                   validation set, the list of
                                                    attributes (np.array),
340
            shape = [num_classes_all, attribute_length].
341
342
            Caution: the attributes must be in the order according to the
343
                                                   labels.
            :param WV_list_all: The relationship of labels and word
344
                                                   vectors in training and
345
346
                                                   validation set, the list of
347
                                                    attributes (np.array),
348
            shape = [num_classes_all, word_vector_length].
            Caution: the attributes must be in the order according to the
349
350
                                                   labels.
351
            :param attri_list_train: The relationship of labels and
352
                                                   attributes in training set,
                                                    the list of attributes (np
353
354
                                                   .arrav).
            shape = [num_classes_train, attribute_length].
355
356
            Caution: the attributes must be in the order according to the
357
                                                   labels.
            :param WV_list_train: The relationship of labels and word
358
359
                                                   vectors in training set,
                                                   the list of attributes (np.
360
361
                                                   array),
            shape = [num_classes_train, word_vector_length].
362
            Caution: the attributes must be in the order according to the
363
364
            :param vgg19\_npy\_path: The path of "vgg19.npy".
365
            :param trainable: A bool tensor, whether the {\it VGG} is trainable.
366
            : \verb"param img_shape: The width or height of image, int.
367
368
            :param regul_DA: The rate of direct attribute learning
                                                   regularization, positive
369
370
                                                   float.
371
            :param regul_IW: The rate of indirect work vector learning
                                                   regularization, positive
372
373
                                                   float.
374
            relu = tf.nn.relu
375
            tanh = tf.nn.tanh
376
377
            sigmoid = tf.nn.sigmoid
            BatchNormalization = tf.layers.batch_normalization
378
```

```
dropout = tf.nn.dropout
379
            dense = tf.layers.dense
380
            softmax = tf.nn.softmax
381
382
            if vgg19_npy_path is not None:
383
                 self.data_dict = np.load(vgg19_npy_path, encoding='latin1'
384
385
                                                       ).item()
386
            else:
                 self.data_dict = None
387
388
389
            self.var_dict = {}
            self.trainable = trainable
390
            self.learning_rate = learning_rate
391
392
            self.img_width = img_shape
393
            self.img_height = img_shape
394
            self.attribute_length = attribute_length
395
            self.num_classes_all = num_classes_all
396
            self.num_classes_train = num_classes_train
397
398
            self.attri_list_all = attri_list_all
            self.attri_list_train = attri_list_train
399
            self.WV_list_train = WV_list_train
400
            self.WV_list_all = WV_list_all
401
402
            self.img_tensor = tf.placeholder(tf.float32,
403
404
                 shape=[None, self.img_width, self.img_height, 3])
            self.img_attribute = tf.placeholder(tf.float32,
405
                 shape=[None, self.attribute_length])
406
407
            self.img_label_all = tf.placeholder(tf.float32,
                 shape=[None, self.num_classes_all])
408
            self.img_label_train = tf.placeholder(tf.float32,
409
410
                 shape=[None, self.num_classes_train])
            self.dropout = tf.placeholder(tf.float32)
411
412
            self.regul_DA = max(0., regul_DA)
413
            self.regul_IW = max(0., regul_IW)
414
415
            self.build(input_image=self.img_tensor, include_top=False)
416
            self.feature = tf.reshape(self.pool5, [-1, 25088])
417
418
            h_fc1 = dropout(sigmoid(BatchNormalization(
419
                 dense(self.feature, 1024), axis=1, training=True)),
420
421
            keep_prob=1-self.dropout)
422
            # Direct Attribute Learning.
423
424
            self.predic_attr_ = dense(h_fc1, self.attribute_length)
425
            self.predic_attr = sigmoid(self.predic_attr_)
426
427
            # Indirect WordVec learning.
428
            self.predic_label_train_ = dense(
429
430
                 self.feature, self.num_classes_train)
431
            self.predic_label_train = softmax(self.predic_label_train_)
432
433
            self.predic_WV = self.Get_WV(
434
                 self.predic_label_train, self.WV_list_train)
435
436
            # predict label.
437
438
            self.predic_label = self.Get_label(
439
                 self.predic_attr, self.predic_WV, self.attri_list_all,
                                                       self.WV_list_all)
440
441
442
            self.acc = self.acc_label(self.img_label_all, self.
443
                                                   predic_label)
```

```
444
            self.loss = (1.0 / (1.0 + self.regul_DA + self.regul_IW)) * tf
445
446
                                                   .reduce_mean(tf.nn.
                                                   sigmoid_cross_entropy_with_logits
447
                                                   (labels=self.img_attribute,
448
                                                    logits=self.predic_attr_))
449
450
                                                    + (self.regul_DA / (1.0 +
                                                   self.regul_DA + self.
451
                                                   regul_I\bar{W})) * tf.reduce_mean
452
453
                                                   ((self.img_label_all - self
454
                                                   .predic_label) ** 2) + (
                                                   self.regul_IW / (1.0 + self
455
                                                   .regul_DA + self.regul_IW))
456
                                                    * tf.reduce_mean(tf.nn.
457
458
                                                   softmax_cross_entropy_with_logits
459
                                                   (labels=self.
460
                                                   img_label_train, logits=
                                                   self.predic_label_train_))
461
462
463
            self.train_op = tf.train.AdamOptimizer(learning_rate=self.
464
                                                   learning_rate).minimize(
                                                   self.loss)
465
466
        def build(self, input_image, include_top=False, train_mode=None):
467
468
469
            Load variable from .npy file to build the VGG19.
            :param input_image: RGB image tensor: [batch, height, width, 3
470
                                                   ]. Values scaled [0, 1].
471
472
            :param include_top: A bool tensor, whether to include the
473
                                                   fully connected layers.
            :param train_mode: A bool tensor, usually a placeholder: if
474
                                                   True, dropout will be
475
                                                   turned on
476
            .....
477
478
            VGG_MEAN = [103.939, 116.779, 123.68]
479
480
481
            input_image_scaled = input_image * 255.0
482
            # Convert RGB to BGR.
483
            red, green, blue = tf.split(axis=3, num_or_size_splits=3,
484
                                                   value=input_image_scaled)
485
486
            assert red.get_shape().as_list()[1:] == [self.img_width, self.
487
                                                   img_height, 1]
488
489
            assert green.get_shape().as_list()[1:] == [self.img_width,
                                                   self.img_height, 1]
490
            assert blue.get_shape().as_list()[1:] == [self.img_width, self
491
                                                   .img_height, 1]
492
493
            bgr_image = tf.concat(axis=3, values=[
494
495
                 blue - VGG_MEAN[0],
                 green - VGG_MEAN[1],
496
                 red - VGG_MEAN[2],
497
            ])
498
            assert bgr_image.get_shape().as_list()[1:] == [224, 224, 3]
499
500
501
            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"
502
503
504
            self.pool1 = self.max_pool(self.conv1_2, 'pool1')
505
            self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
506
507
            self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128, "
                                                   conv2_2")
508
```

```
self.pool2 = self.max_pool(self.conv2_2, 'pool2')
509
510
            self.conv3_1 = self.conv_layer(self.pool2, 128, 256, "conv3_1"
511
512
            self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256, "
513
                                                   conv3_2")
514
            self.conv3_3 = self.conv_layer(self.conv3_2, 256, 256, "
515
                                                   conv3_3")
516
            self.conv3_4 = self.conv_layer(self.conv3_3, 256, 256, "
517
                                                   conv3_4")
518
519
            self.pool3 = self.max_pool(self.conv3_4, 'pool3')
520
            self.conv4_1 = self.conv_layer(self.pool3, 256, 512, "conv4_1"
521
522
            self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512, "
523
                                                   conv4_2")
524
            self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512, "
525
                                                   conv4_3")
526
            self.conv4_4 = self.conv_layer(self.conv4_3, 512, 512, "
527
                                                   conv4_4")
528
            self.pool4 = self.max_pool(self.conv4_4, 'pool4')
529
530
531
            self.conv5_1 = self.conv_layer(self.pool4, 512, 512, "conv5_1"
532
            self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512, "
533
                                                   conv5_2")
534
            self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512, "
535
                                                   conv5_3")
536
            self.conv5_4 = self.conv_layer(self.conv5_3, 512, 512, "
537
                                                   conv5_4")
538
            self.pool5 = self.max_pool(self.conv5_4, 'pool5')
539
540
541
            if include_top:
                 self.fc6 = self.fc_layer(self.pool5, 25088, 4096, "fc6")
542
                                                       # 25088 = ((224 // (2
543
                                                       ** 5)) ** 2) * 512
544
545
                 self.relu6 = tf.nn.relu(self.fc6)
546
                 if train_mode is not None:
                     self.relu6 = tf.cond(train_mode, lambda: tf.nn.dropout
547
                                                           (self.relu6, self.
548
                                                           dropout), lambda:
549
                                                           self.relu6)
550
                 elif self.trainable:
551
                     self.relu6 = tf.nn.dropout(self.relu6, self.dropout)
552
553
554
                 self.fc7 = self.fc_layer(self.relu6, 4096, 4096, "fc7")
                 self.relu7 = tf.nn.relu(self.fc7)
555
                 if train_mode is not None:
556
                     self.relu7 = tf.cond(train_mode, lambda: tf.nn.dropout
557
                                                            (self.relu7, self.
558
                                                           dropout), lambda:
559
560
                                                           self.relu7)
                 elif self.trainable:
561
                     self.relu7 = tf.nn.dropout(self.relu7, self.dropout)
562
563
                 self.fc8 = self.fc_layer(self.relu7, 4096, 1000, "fc8")
564
565
566
                 self.prob = tf.nn.softmax(self.fc8, name="prob")
567
568
            self.data_dict = None
569
            return None
579
```

A.2 DAP

572

```
573
    class DirectAttribute_VGG19(object):
        def __init__(self, learning_rate, attribute_length,
575
                                              num_classes_all, attri_list_all
576
577
                      vgg19_npy_path=None,
578
                      trainable=False,
579
580
                      img\_shape=224,
                      regul=0.0):
581
582
583
            :param learning_rate: Learning rate, float.
             :param attribute_length: The length of attribute, int.
584
            :param num_classes_all: The number of classes in training and
585
                                                   validation set, int.
586
            :param attri_list_all: The relationship of labels and
587
588
                                                   attributes, the list of
589
                                                   attributes (np.array),
            shape = [num_classes, attribute_length].
590
591
            Caution: the attributes must be in the order according to the
592
            :param vgg19_npy_path: The path of "vgg19.npy".
593
            :param trainable: A bool tensor, whether the VGG is trainable.
594
            :param img_shape: The width or height of image, int.
595
            :param regul: The rate of regularization, positive float.
596
597
            relu = tf.nn.relu
598
            tanh = tf.nn.tanh
599
600
            sigmoid = tf.nn.sigmoid
601
            BatchNormalization = tf.layers.batch_normalization
602
            dropout = tf.nn.dropout
603
            dense = tf.layers.dense
604
            if vgg19_npy_path is not None:
605
606
                 self.data_dict = np.load(vgg19_npy_path, encoding='latin1'
                                                       ).item()
607
            else:
608
609
                 self.data_dict = None
610
            self.var_dict = {}
611
            self.trainable = trainable
612
            self.learning_rate = learning_rate
613
614
615
            self.img_width = img_shape
            self.img_height = img_shape
616
            self.attribute_length = attribute_length
617
            self.num_classes_all = num_classes_all
618
619
            self.attri_list_all = attri_list_all
620
            self.img_tensor = tf.placeholder(tf.float32, shape=[None, self
621
                                                   .img_width, self.img_height
622
                                                    3])
623
            self.img_attribute = tf.placeholder(tf.float32, shape=[None,
624
625
                                                   self.attribute_length])
626
            self.img_label_all = tf.placeholder(tf.float32, shape=[None,
                                                   self.num_classes_all])
627
            self.dropout = tf.placeholder(tf.float32)
628
            self.regul = max(0., regul)
629
630
            self.build(input_image=self.img_tensor, include_top=False)
631
632
            self.feature = tf.reshape(self.pool5, [-1, 25088])
633
634
635
            h_fc1 = dropout(sigmoid(BatchNormalization(dense(self.feature,
                                                    1024), axis=1, training=
636
```

```
True)), keep_prob=1 - self.
637
                                                   dropout)
638
639
            self.predic_attr_ = dense(h_fc1, self.attribute_length)
640
641
            self.predic_attr = sigmoid(self.predic_attr_)
642
643
644
            self.predic_label = self.Get_label(self.predic_attr, self.
645
                                                   attri_list_all)
646
647
648
            self.acc = self.acc_label(self.img_label_all, self.
                                                   predic_label)
649
650
            self.loss = (1.0/(1.0+self.regul)) * tf.reduce_mean(tf.nn.
651
652
                                                   sigmoid_cross_entropy_with_logits
                                                   (labels=self.img_attribute,
653
                                                    logits=self.predic_attr_))
654
                                                    + (self.regul / (1.0 +
655
656
                                                   self.regul)) * tf.
                                                   reduce_mean((self.
657
                                                   img_label_all-self.
658
659
                                                   predic_label) ** 2)
660
            self.train_op = tf.train.AdamOptimizer(learning_rate=self.
661
662
                                                   learning_rate).minimize(
                                                   self.loss)
663
664
665
        def build(self, input_image, include_top=False, train_mode=None):
666
            Load variable from .npy file to build the VGG19.
667
            :param input_image: RGB image tensor: [batch, height, width, 3
668
                                                   ]. Values scaled [0, 1].
669
            :param include_top: A bool tensor, whether to include the
670
671
                                                   fully connected layers.
672
            :param train_mode: A bool tensor, usually a placeholder: if
673
                                                   True, dropout will be
674
                                                   turned on
            .....
675
676
            VGG_MEAN = [103.939, 116.779, 123.68]
677
678
            input_image_scaled = input_image * 255.0
679
680
            red, green, blue = tf.split(axis=3, num_or_size_splits=3,
681
682
                                                   value=input_image_scaled)
683
            assert red.get_shape().as_list()[1:] == [self.img_width, self.
684
                                                   img_height, 1]
685
            assert green.get_shape().as_list()[1:] == [self.img_width,
686
                                                   self.img_height, 1]
687
688
            assert blue.get_shape().as_list()[1:] == [self.img_width, self
                                                   .img_height, 1]
689
690
691
            bgr_image = tf.concat(axis=3, values=[
                 blue - VGG_MEAN[0]
692
                 green - VGG_MEAN[1],
693
694
                 red - VGG_MEAN[2],
            ])
695
696
            assert bgr_image.get_shape().as_list()[1:] == [224, 224, 3]
697
            self.conv1_1 = self.conv_layer(bgr_image, 3, 64, "conv1_1")
698
699
            self.conv1_2 = self.conv_layer(self.conv1_1, 64, 64, "conv1_2"
700
701
            self.pool1 = self.max_pool(self.conv1_2, 'pool1')
```

```
702
            self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
703
            self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128, "
704
                                                   conv2_2")
705
            self.pool2 = self.max_pool(self.conv2_2, 'pool2')
706
707
            self.conv3_1 = self.conv_layer(self.pool2, 128, 256, "conv3_1"
708
709
            self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256, "
710
                                                   conv3_2")
711
712
            self.conv3_3 = self.conv_layer(self.conv3_2,
                                                             256, 256, "
                                                   conv3_3")
713
            self.conv3_4 = self.conv_layer(self.conv3_3, 256, 256, "
714
715
                                                   conv3_4")
716
            self.pool3 = self.max_pool(self.conv3_4, 'pool3')
717
            self.conv4_1 = self.conv_layer(self.pool3, 256, 512, "conv4_1")
718
719
            self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512, "
720
                                                   conv4_2")
721
            self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512, "
722
                                                   conv4 3")
723
724
            self.conv4_4 = self.conv_layer(self.conv4_3, 512, 512, "
                                                   conv4_4")
725
            self.pool4 = self.max_pool(self.conv4_4, 'pool4')
726
727
            self.conv5_1 = self.conv_layer(self.pool4, 512, 512, "conv5_1"
728
729
730
            self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512, "
                                                   conv5_2")
731
            self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512, "
732
                                                   conv5_3")
733
            self.conv5_4 = self.conv_layer(self.conv5_3, 512, 512, "
734
                                                   conv5_4")
735
            self.pool5 = self.max_pool(self.conv5_4, 'pool5')
736
737
738
            if include_top:
                 self.fc6 = self.fc_layer(self.pool5, 25088, 4096, "fc6")
739
                                                       # 25088 = ((224 // (2
740
                                                       ** 5)) ** 2) * 512
741
                 self.relu6 = tf.nn.relu(self.fc6)
742
                 if train_mode is not None:
743
                     self.relu6 = tf.cond(train_mode, lambda: tf.nn.dropout
744
                                                            (self.relu6, self.
745
                                                           dropout), lambda:
746
747
                                                           self.relu6)
                 elif self.trainable:
748
                     self.relu6 = tf.nn.dropout(self.relu6, self.dropout)
749
750
                 self.fc7 = self.fc_layer(self.relu6, 4096, 4096, "fc7")
751
                 self.relu7 = tf.nn.relu(self.fc7)
752
753
                 if train_mode is not None:
                     self.relu7 = tf.cond(train_mode, lambda: tf.nn.dropout
754
                                                            (self.relu7, self.
755
756
                                                            dropout), lambda:
757
                                                           self.relu7)
                 elif self.trainable:
758
759
                     self.relu7 = tf.nn.dropout(self.relu7, self.dropout)
760
                 self.fc8 = self.fc_layer(self.relu7, 4096, 1000, "fc8")
761
762
                 self.prob = tf.nn.softmax(self.fc8, name="prob")
763
764
765
            self.data_dict = None
766
```

A.3 IAP

```
class IndirectAttribute_VGG19(object):
772
        def __init__(self, learning_rate, attribute_length,
773
                                              num_classes_all,
                                              num_classes_train,
774
775
                                              attri_list_all,
776
                                              attri_list_train,
777
                      vgg19_npy_path=None,
                      trainable=False,
778
                      img_shape=224,
779
                      regul=0.0):
780
781
            :param learning_rate: Learning rate, float.
782
            :param attribute_length: The length of attribute, int.
783
784
             :param num_classes_all: The number of classes in training and
785
                                                   validation set, int.
786
            :param num_classes_train: The number of classes in training
                                                   set, int.
787
            :param attri_list_all: The relationship of labels and
788
                                                   attributes in training and
789
790
                                                   validation set, the list of
                                                    attributes (np.array),
791
            shape = [num_classes_all, attribute_length].
792
793
            Caution: the attributes must be in the order according to the
794
                                                   labels.
            :param attri_list_train: The relationship of labels and
795
796
                                                   attributes in training set,
797
                                                    the list of attributes (np
798
                                                   .array),
799
            shape = [num_classes_train, attribute_length].
            Caution: the attributes must be in the order according to the
800
801
                                                   labels.
802
            :param vgg19_npy_path: The path of "vgg19.npy".
            :param trainable: A bool tensor, whether the VGG is trainable.
803
            :param img_shape: The width or height of image, int.
804
            :param regul: The rate of regularization, positive float.
805
806
807
            relu = tf.nn.relu
808
            tanh = tf.nn.tanh
            sigmoid = tf.nn.sigmoid
809
810
            BatchNormalization = tf.layers.batch_normalization
            dropout = tf.nn.dropout
811
            softmax = tf.nn.softmax
812
            dense = tf.layers.dense
813
814
            self.img_width = img_shape
815
            self.img_height = img_shape
816
            self.attribute_length = attribute_length
817
            self.num_classes_all = num_classes_all
818
819
            self.num_classes_train = num_classes_train
            self.attri_list_all = attri_list_all
820
            self.attri_list_train = attri_list_train
821
822
            if vgg19_npy_path is not None:
823
                 self.data_dict = np.load(vgg19_npy_path, encoding='latin1'
824
825
                                                       ).item()
            else:
826
                 self.data_dict = None
827
828
            self.var_dict = {}
829
```

```
self.trainable = trainable
830
            self.learning_rate = learning_rate
831
            self.regul = max(0., regul)
832
833
            self.img_tensor = tf.placeholder(tf.float32, shape=[None, self
834
                                                   .img_width, self.img_height
835
836
                                                    3])
            self.img_attribute = tf.placeholder(tf.float32, shape=[None,
837
                                                   self.attribute_length])
838
            self.img_label_all = tf.placeholder(tf.float32, shape=[None,
839
840
                                                   self.num_classes_all])
            self.img_label_train = tf.placeholder(tf.float32, shape=[None,
841
                                                    self.num_classes_train])
842
843
            self.dropout = tf.placeholder(tf.float32)
844
            self.build(input_image=self.img_tensor, include_top=False)
845
846
            self.feature = tf.reshape(self.pool5, [-1, 25088])
847
848
849
            h_fc1 = dropout(sigmoid(BatchNormalization(dense(self.feature,
                                                    1024), axis=1, training=
850
                                                   True)), keep_prob=1 - self.
851
852
                                                   dropout)
853
854
            self.predic_label_train_ = dense(h_fc1, self.num_classes_train
855
856
            self.predic_label_train = softmax(self.predic_label_train_)
857
858
            self.predic_attr = self.Get_attr(self.predic_label_train, self
859
                                                   .attri_list_train)
860
861
            self.predic_label = self.Get_label(self.predic_attr, self.
862
863
                                                   attri_list_all)
864
865
            self.acc = self.acc_label(self.img_label_all, self.
866
                                                   predic_label)
867
            self.loss = (1.0/(1.0+self.regul)) * tf.reduce_mean(tf.nn.
868
                                                   \verb|softmax_cross_entropy_with_l| og its \\
869
870
                                                   (labels=self.
                                                   img_label_train, logits=
871
                                                   self.predic_label_train_))
872
                                                   + (self.regul / (1.0 + self
873
                                                   .regul)) * tf.reduce_mean(
874
875
                                                   tf.nn.
                                                   sigmoid_cross_entropy_with_logits
876
                                                   (labels=self.img_attribute,
877
                                                    logits=self.predic_attr))
878
879
            self.train_op = tf.train.AdamOptimizer(learning_rate=self.
880
881
                                                   learning_rate).minimize(
                                                   self.loss)
882
883
884
        def build(self, input_image, include_top=False, train_mode=None):
885
            Load variable from .npy file to build the VGG19.
886
887
            :param input_image: RGB image tensor: [batch, height, width, 3
                                                   ]. Values scaled [0, 1].
888
889
            :param include_top: A bool tensor, whether to include the
890
                                                   fully connected layers.
            :param train_mode: A bool tensor, usually a placeholder: if
891
892
                                                   True, dropout will be
893
                                                   turned on
894
```

```
895
            VGG_MEAN = [103.939, 116.779, 123.68]
896
897
            input_image_scaled = input_image * 255.0
898
899
            red, green, blue = tf.split(axis=3, num_or_size_splits=3,
900
901
                                                  value=input_image_scaled)
902
            assert red.get_shape().as_list()[1:] == [self.img_width, self.
903
                                                  img_height, 1]
904
905
            assert green.get_shape().as_list()[1:] == [self.img_width,
906
                                                   self.img_height, 1]
            assert blue.get_shape().as_list()[1:] == [self.img_width, self
907
                                                   .img_height, 1]
908
909
            bgr_image = tf.concat(axis=3, values=[
910
                 blue - VGG_MEAN[0],
911
                 green - VGG_MEAN[1],
912
                 red - VGG_MEAN[2],
913
            ])
914
            assert bgr_image.get_shape().as_list()[1:] == [224, 224, 3]
915
916
917
            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"
918
919
920
            self.pool1 = self.max_pool(self.conv1_2, 'pool1')
921
922
            self.conv2_1 = self.conv_layer(self.pool1, 64, 128, "conv2_1")
923
            self.conv2_2 = self.conv_layer(self.conv2_1, 128, 128, "
                                                  conv2_2")
924
            self.pool2 = self.max_pool(self.conv2_2, 'pool2')
925
926
            self.conv3_1 = self.conv_layer(self.pool2, 128, 256, "conv3_1"
927
928
            self.conv3_2 = self.conv_layer(self.conv3_1, 256, 256, "
929
                                                  conv3_2")
930
931
            self.conv3_3 = self.conv_layer(self.conv3_2, 256, 256, "
932
                                                  conv3_3")
            self.conv3_4 = self.conv_layer(self.conv3_3, 256, 256, "
933
                                                   conv3_4")
934
            self.pool3 = self.max_pool(self.conv3_4, 'pool3')
935
936
            self.conv4_1 = self.conv_layer(self.pool3, 256, 512, "conv4_1"
937
938
            self.conv4_2 = self.conv_layer(self.conv4_1, 512, 512, "
939
940
                                                  conv4_2")
            self.conv4_3 = self.conv_layer(self.conv4_2, 512, 512, "
941
                                                  conv4_3")
942
            self.conv4_4 = self.conv_layer(self.conv4_3, 512, 512, "
943
                                                  conv4_4")
944
            self.pool4 = self.max_pool(self.conv4_4, 'pool4')
945
946
            self.conv5_1 = self.conv_layer(self.pool4, 512, 512, "conv5_1"
947
948
949
            self.conv5_2 = self.conv_layer(self.conv5_1, 512, 512, "
950
                                                  conv5_2")
            self.conv5_3 = self.conv_layer(self.conv5_2, 512, 512, "
951
952
                                                  conv5_3")
            self.conv5_4 = self.conv_layer(self.conv5_3, 512, 512, "
953
                                                  conv5_4")
954
955
            self.pool5 = self.max_pool(self.conv5_4, 'pool5')
956
            if include_top:
957
958
                 self.fc6 = self.fc_layer(self.pool5, 25088, 4096, "fc6")
                 self.relu6 = tf.nn.relu(self.fc6)
959
```

```
if train_mode is not None:
960
                     self.relu6 = tf.cond(train_mode, lambda: tf.nn.dropout
961
                                                            (self.relu6, self.
962
                                                            dropout), lambda:
963
                                                            self.relu6)
964
                 elif self.trainable:
965
                     self.relu6 = tf.nn.dropout(self.relu6, self.dropout)
966
967
                 self.fc7 = self.fc_layer(self.relu6, 4096, 4096, "fc7")
968
969
                 self.relu7 = tf.nn.relu(self.fc7)
970
                 if train_mode is not None:
                     self.relu7 = tf.cond(train_mode, lambda: tf.nn.dropout
971
                                                            (self.relu7, self.
972
                                                            dropout), lambda:
973
                                                            self.relu7)
974
975
                 elif self.trainable:
                     self.relu7 = tf.nn.dropout(self.relu7, self.dropout)
976
977
                 self.fc8 = self.fc_layer(self.relu7, 4096, self.
978
979
                                                        num_classes_train, "fc8
                                                        ")
980
981
982
                 self.prob = tf.nn.softmax(self.fc8, name="prob")
983
            self.data_dict = None
984
985
            return None
989
```

988 A.4 Data Pre-processing

```
989
    def Get_next_batch(Train_or_Vali, batch_size, epoch, use_word2vec=0,
990
991
                                          attribute = 'conti'):
992
       Get next batch according to size.
993
994
       :param Train_or_Vali: Whether to train or validate the model.
995
       :param batch_size: The size of the training batch.
       :param use_word2vec: If use the word2vec feature as an attribute
996
       :return: training data, shape = [batch_size, img_width, img_height,
997
                                            img_path],
998
       attribute, shape = [batch_size, attribute_length]
999
       and label(one-hot), shape = [batch_size, num_classes]
1000
      for next batch, type = np.array.
1001
1002
1003
       img_batch_tensor = np.zeros((batch_size, img_width, img_height, 3))
       if not use_word2vec:
1004
1005
        attribute_batch_tensor = np.ndarray((batch_size, 85))
1006
       else:
        attribute_batch_tensor = np.ndarray((batch_size, 85 + use_word2vec
1007
1008
       # if not use_all_label:
1009
      #
          label_batch_tensor = np.ndarray((batch_size, 20))
1010
1011
1012
       label_batch_tensor = np.zeros((batch_size, 25))
1013
1014
      if Train_or_Vali == 'train':
1015
         img_path = './Data/train_zsl/'
1016
         for batch in range(batch_size):
1017
1018
           file_name = train_file_list[(batch_size*epoch + batch)%
                                                 train_file_len]
1019
           img_batch_tensor[batch] = image2tensor(img_path + file_name)
1020
1021
           label_name = file_name.split(',')[0]
           if attribute == 'bi':
1022
```

```
if not use_word2vec:
1023
1024
               attribute_batch_tensor[batch] = attribute_bi_dict[label_name
1025
1026
1027
               attribute_batch_tensor[batch][:85] = attribute_bi_dict[
                                                      label_name]
1028
               attribute_batch_tensor[batch][85:] = glove(label_name,
1029
                                                      use_word2vec)
1030
           elif attribute == 'conti':
1031
1032
             if not use_word2vec:
1033
               attribute_batch_tensor[batch] = attribute_conti_dict[
                                                      label_name]
1034
             else:
1035
1036
               attribute_batch_tensor[batch][:85] = attribute_conti_dict[
                                                      label_name]
1037
               attribute_batch_tensor[batch][85:] = glove(label_name,
1038
                                                      use_word2vec)
1039
           label_batch_tensor[batch][classes_new_dict[label_name]] = 1
1040
1041
       elif Train_or_Vali == 'validation':
1042
         img_path = './Data/validation_zsl/'
1043
         for batch in range(batch_size):
1044
           file_name = vali_file_list[(batch_size*epoch + batch)%
1045
                                                 vali_file_len]
1046
           img_batch_tensor[batch] = image2tensor(img_path + file_name)
1047
           label_name = file_name.split('_')[0]
1048
           if attribute == 'bi':
1049
             if not use_word2vec:
1050
               attribute_batch_tensor[batch] = attribute_bi_dict[label_name
1051
1052
1053
               attribute_batch_tensor[batch][:85] = attribute_bi_dict[
1054
                                                      label_name]
1055
               attribute_batch_tensor[batch][85:] = glove(label_name,
1056
                                                      use_word2vec)
1057
           elif attribute == 'conti':
1058
1059
             if not use_word2vec:
               attribute_batch_tensor[batch] = attribute_conti_dict[
1060
                                                      label_name]
1061
             else:
1062
               attribute_batch_tensor[batch][:85] = attribute_conti_dict[
1063
1064
                                                      label_name]
               attribute_batch_tensor[batch][85:] = glove(label_name,
1065
                                                      use_word2vec)
1066
1067
           label_batch_tensor[batch][classes_new_dict[label_name]] = 1
1068
       else:
         print("ERROR!You should input train or validation")
1069
1070
1071
       return img_batch_tensor.astype('float32'), attribute_batch_tensor.
                                             astype('float32'),
1072
1073
                                             label_batch_tensor.astype('
                                             float32')
1874
```