Yelp Restaurant Photo Classification

Yu Wen, Qian Shen, Huijing Zhang, Shiyu Zhang

Group ID: 4
Group Name: Data Dogs

27/4/2016

abstract

Can we build a model to automatically tag restaurant with multiple labels using a dataset of user-submitted photos? Currently, restaurant labels are manually selected by Yelp users when they submit a review. Selecting the labels is optional, leaving some restaurants un- or only partially-categorized. Yelp needs researchers to design an efficient algorithm to label those images automatically. Our group investigate this question using a dataset from yelp and try to implement SVM and CNN algorithms to it. However, how CNN or SVM best copies with multi-label images still remains an open question, mainly due to the complex underlying object layouts and insufficient multi-label training images. Experimental results of the datasets well demonstrate the success of our implementation. Finally, the accuracy of our classification reaches 80.41% while the top 1 researcher reaches 83.177%.

1 Introduction

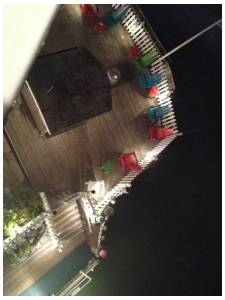
Most people rely on Yelp to locate great restaurants, write reviews, and upload restaurant photos to Yelp via their mobile devices. Yelp is now hosting tens of millions of photos shared by Yelpers all over the world. Yelp aims to add attribute labels for each restaurant photo shared by Yelpers, classifying restaurants into different categories. By labeling restaurants with distinct business attributes, Yelp is able to translate uploaded photos into more explicit category information for each restaurant. There are 9 different business attributes listed, 1) good for lunch, 2) good for dinner, 3) takes reservations, 4) outdoor seating, 5) restaurant is expensive, 6) has alcohol, 7) ambience is classy, 8) has table service, 9) good for kids. Currently, these restaurant labels can only be manually selected by Yelp users while submitting a review. Since this selection is optional, many uploaded restaurant photos are not or partially classified. To Yelp users, when looking for certain category restaurants, these restaurant labels will help them quickly find out the ones that satisfy their requirement. For example, Sam would like to treat his parents a great dinner. He will be interested in restaurants whose attribute label is good for dinner. With the help with restaurant category label, he is able to quickly find out desirable restaurants.

(a) Good for lunch

(b) Good for dinner

(c) takes reservations

(d) outdoor seating



(e) restaurant is expensive

(f) has alcohol

(g) ambience is classy

(h) has table service

(i) good for kids

Figure 1: 9 kinds of labels

There is no doubt that, classifying restaurants into different categories can better serve users' request in a more efficient way. Uploaded restaurant photos themselves contain great information which could be provided to users, helping them make a better decision of whether choosing a restaurant or not. Relying on only Yelp users to label restaurants is undesirable, since some users may forget to select the attribute or are willing to select at all. According to Yelp researchers, there are only a small number of users who would like to category restaurant photos uploaded. However, the accuracy of manually selected features is not reliable. It is entirely possible that, users select a feature at will without careful consideration because of limited time or other issues. However, appropriate classification of these restaurant photos plays a significant role from the website's side perspective. Lacking thorough analysis of attributes for each restaurant photos will lead to poor website performance. In a worse case, users may abandon Yelp if all the restaurants are listed without providing detailed and accurate category information, or their categories are presented in a mass. Thus, the more accurate classification information we can mine from these photos the better.

In this project, we would like to get rid of manually labeling from Yelp users. Instead, develop a data mining technique to build a restaurant photo classifier. This classifier is responsible for taking a restaurant photo as input, and automatically attaching some of the above mentioned 9 business attributes to it based on the algorithm developed. Our expectation is that, after building the classifier using training data, the classifier is able to predict business attribute for each restaurant photo in an accurate way.

With the help of this classifier, Yelp can gain a more detailed and precise analysis over these uploaded restaurant photos and enhance the quality of service.

2 Problem definition and formalization

In this project, the model we would like to build takes a bunch of images for one specific restaurant and make predictions of the 9 business attributes mentioned above. For example, there are 233 images of restaurant MexicanRolls. Then we input these 233 images into our model and get the set of business attributes of MexicanRolls as 1,3,5,8, which indicates that has the attributes 1) good for lunch, 3) takes reservations, 5) restaurant is expensive 8) has table service. Note that, for one specific restaurant, the output may contain 0 to 9 attributes. So, this problem is a multi?label classification problem. At the very beginning, it is easy to come out with the idea that we can solve it by convolution neural network in deep learning. Otherwise, SVM is also a typical solution to the multi-label classification problem. So we decided to get started with a relative easy algorithm, SVM. SVM are very attractive for the classification of remotely sensed data[1]. This approach seeks to find optimal separating hyperplane between classes distributions, the support vectors, with the other training cases effectively discarded. The basic nature of classification with an SVM can be illustrated most easily for the simple situation in which there are two linearly separable classes in q-dimensional space[3]. After extracting features from images, we can simply use SVM classify the image to different labels.

Until the emergence of ImageNet dataset, there was almost no publicly available large-scale benchmark data for image classification. This is mostly because class labels are expensive to obtain. A key challenge for us here is how to achieve efficiency in both feature extraction and classifier training without compromising performance[2]. We consider about using CaffeNet or Theano, we need to evaluate the result of them and choose one to move further. Figure 2 shows the work flow of hypothesis extraction by CNN[2]. Besides, CNN framework, it still exists some ways to extract features, HOG is one of them. The Histogram of Oriented Gradient(HOG) feature descriptor [8] is popular for object detection. Our group will also use this method to get information from photos.

3 Data description and preprocessing

3.1 Data description In this project, we use the data given by yelp which contains 234,843 photos taken from 2,000 restaurants. Details are as followings:

1. train.csv

   It continas a field of "business_id", which indicates a restaurant, and a field of "labels". There are 2000 examples in this file. Here are two examples:

bussiness_id	labels
1000	1 2 3 4 5 6 7
1031	6 8

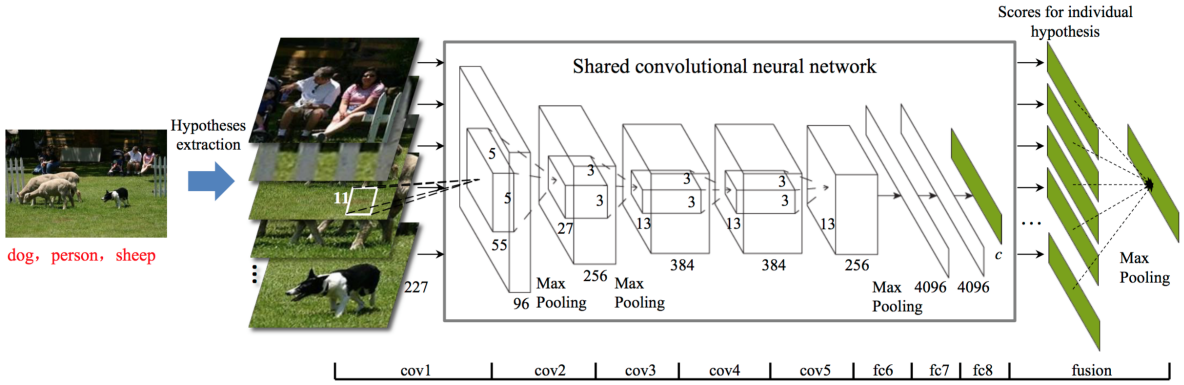


Figure 2: **An illustration of the infrastructure of the proposed HCP.**

2. train_photo

   There is a folder contains photos of the training set. There are 234,843 photos of restaurants, each photo has a photo_id as their file name.

3. train_photo_to_biz_ids.csv

   This file maps the train_photo id to business id, which gave us information about for a business_id in train.csv, which images in train_photo are taken from this restaurant.

4. test_photo

   There is a folder contains photos of the test set. There are 7GB photos of restaurants, each photo has a photo_id.

5. test_photo_to_biz_ids.csv

   It maps the test_photo id to business id, similar to train_photo_to_biz_ids.csv.

   Note that this projects derived from a Kaggle competition, so there are no true label of restaurants in test_photo_to_biz_ids.csv. If we want to see the performance of our model on test_photo_to_biz_ids.csv, we have to hand in the prediction results on Kaggle, then , we can see the evaluation given by F1 score.

   Alternatively, we have randomly sampled 20% data from train.csv to be our development set leaving the 80% as our training set.

3.2 data preprocessing Since for one restaurant, there could be a bundle of images which consists of one training example, we can use two kind of methods of preprocessing.The first method is to preprocess every image as a single vector and then set labels of every business id by using ?union? or ?and? of labels produced by predicting on each image. The second method is to ?combine? all images under a single business id to obtain one feature vector.

We have tried several ways to preprocess our data:

3.2.1 Resizing Training images have different resolutions, as a result, we need to resize the images to 256*256 in advance to ensure each image can have same number of pixels. In this way, this resizing make sure we have the unified input dimension if we use SVM, feed?forward ANN or CNN to build our model and treat each pixel as a feature of training example. We may benefit from resizing images in a parallel fashion using mapreduce. But after we judge and weight the method, we hope to make things simple, so we choose to use shell commands, something like:

```
for name in /path/to/image/*.JPEG; do
   convert -resize 256*256\! $name $name
done
```

3.2.2 Computing Image Mean This is a way of combining all the imageś information as a vector such that we think the model requires to subtract the image mean from each image. Mean of the image means we get the mean value of RGB in each pixel. Meanwhile In practice, subtracting the mean image from a dataset significantly improves classification accuracies. So we have to compute the mean. In the caffeNet, there are tools can help us achieve this.

3.2.3 Transfer images to grey scale Another method is to transfer our image dataset into grey scale first, then for each pixel, we set each pixel to a grey scale value from 0 to 255, it would be easier for computers to compute but maintain the risk of lacking accuracy.

4 Methods description

4.1 Python Module

1. Comparison between Theano and Caffe

   Theano gives a comprehensive control over Neural Network information. Most academic researchers in the field of deep learning reply on Theano, the grand-daddy of deep-learning frameworks, which is written in Python. Theano is a library that handles multidimensional arrays, like Numpy. Used with other libs, it is well suited to data exploration and intended for research.

   Caffe is a well-known and widely used machine vision library that proved Matlabś implementation of fast convolutional nets to C and C++. Caffe is not intended for other deep-learning applications such as text, sound or time series data. Like other frameworks mentioned here, Caffe has chosen Python for its API.

   As we are totally beginner of the deep learning field, Caffe is a great choice for us because we can use some standard algorithm from it.

2. HDF5

   As we have 470,000 train and test images, we decide to import HDF5 module in python. The h5py package is a Pythonic interface to the HDF5 binary data format. It let you store huge amounts of numerical data, and easily manipulate that data from Numpy. For example, you can slice into multi-terabyte datasets stored on disk, as if they were real NumPy arrays. Thousands of datasets can be stored in a single file, categorized and tagged however you want.

3. Numpy

   NumPy is the fundamental package for scientific computing with Python. It contains among other things:

   • a powerful N-dimensional array object

   • sophisticated (broadcasting) functions

   • tools for integrating C/C++ and Fortran code

   • useful linear algebra, Fourier transform, and random number capabilities

4. Skimage

   Skimage is a collection of algorithms for image processing and computer vision. It is dedicated to image processing, and using naively NumPy arrays as image objects.

5. Pandas

   pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

4.2 Implementation of CaffeNet

1. Definition of Layer

   Caffe models and optimization are defined by plain text schema for ease of experimentation[5]. A Caffe layer is the essence of a neural network layer: it takes one or more blobs as input, and yields one or more blobs as output. Layers have two key responsibilities for the operation of the network as a whole: a forward pass that takes the inputs and produces the outputs, and a backward pass that takes the gradient with respect to the output, and computes the gradients with respect to the parameters and to the inputs, which are in turn back-propagated to earlier layers.

   Caffe provides a complete set of layer types including: convolution, pooling, inner products, nonlinearities like rectified linear and logistic, local response normalization, element-wise operations, and losses like softmax and hinge. These are all the types needed for state-of-the-art visual tasks. Coding custom layers requires minimal effort due to the compositional construction of networks[5].

   For instance, a convolutional layer for 20 filters of size 5*5 is defined using the following text:

```
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
  }
}
```

Every model layer is defined in this way, there is a famous model for handwritten digit record recognition in the caffeNet tutorial, it can reach 99% accuracy in less than a minute with GPU training.

2. The Caffe Layer Architecture

   In caffe, the code for a deep model follows its layered and compositional structure for modularity. The Net has layers, and the computations of the Net are delegated to the layers. All deep model computations are framed as layer types like convolution, pooling, nonlinearities, and so on[6]. By encapsulating the details of the operation, the layer provides modularity since it can be implemented once and then instantiated everywhere. To afford this encapsulation the layers must follow a common protocol:

   - Setup and reshape: the layer setup does one-time initialization like loading parameters while reshape handles the input-output configuration of the layer so that the user does not have to do dimension bookkeeping.
   - Forward:compute the output given the output.
   - Backward:compute the gradient of the output with respect to the input and with respect to the parameters. The layer class follows this protocol in its public method definitions. Each layer type is an inherited class that declares these same methods, as in the cuDNN convolution class declaration. The actual implementations are found in a combination of .cpp and .cu files. For example the CuDNNConvolutionLayer class Setup / Reshape and CPU-mode Forward / Backward methods are implemented in CUDNN_CONV_LAYER.CPP, and the GPU-mode Forward / Backward methods are implemented in CUDNN_CONV_LAYER.CU. Note that CUDNNCONVOLUTIONLAYER is a strategy class for ConvolutionLayer, and does not declare the Forward/Backward_cpu variants of Forward / Backward so that it inherits the standard Caffe CPU mode instead. The GPU mode CUDA code is optional for CPU-only layers or prototyping. The GPU mode falls back to CPU implementations with automatic communication between the host and device as needed. All layer classes follow this .cpp + .cu arrangement.

Once implemented, the layer needs to be included in the model schema. This schema is coded in the Protocol Buffer format in the caffe.proto master definition. To include a layer in the schema:

- Register its type in the LayerType enumeration like CONVOLUTION.
- Define a field to hold the configuration parameters of the layer, if any, like the Convolution-Parater field.
- Define the actual layer parameter message, like the ConlutionParameter.

3. Extract Features

   In addition to end-to-end training, Caffe can also be used to extract semantic features from images using a pre-trained network. These features can be used to downstream in other vision tasks with great success. Figure 3[5] displays a two-dimensional embedding of all the ImageNet validation images, colored by a coarse category that they come from. The nice seperation testifies to a successful semantic embedding.

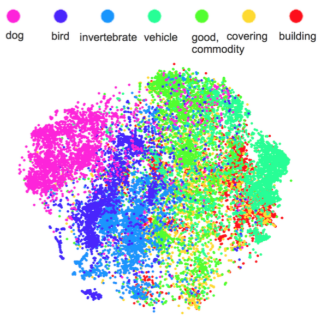


Figure 3: **Features extracted from a deep network, visualized in a 2-dimensional space**

4.3 Implementation of HOG

4.3.1 Definition of HOG HOG is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. This method is similar to that of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, but differs in that it is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy[8].

4.3.2 HOG Algorithm

- Apply an optional global image normalisation to reduce the influence of illumination effects.
- Compute first order image gradients to capture the contour, silhouette and some texture information.

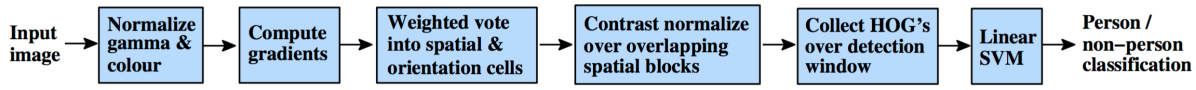


Figure 4: **A feature extraction and object detection chain.**

- Produce an encoding which is sensitive to local image content while remaining resistant to small changes in pose or appearance. This method can pool gradient orientation information locally.
- Compute normalisation which take local groups of cells and contrast normalises their overall responses before passing to next stage.
- The final step collects the HOG descriptors from all blocks of dense overlapping grid of blocks covering the detection window into a combined feature vector for use in the window classifier[8].

Figure 5 is image displayed before and after HOG processing[9].

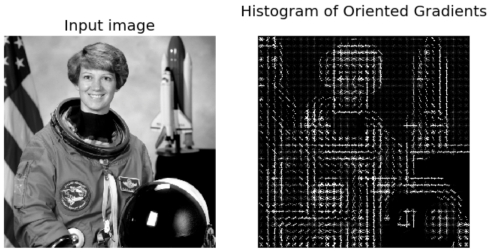


Figure 5: **Features extracted before and after HOG implementation**

5 Experiments design and Evaluation

5.1 Implement CaffeNet

5.2 Implement HOG

- Transfer image from RGB to Gray, resize all images to 256*256 size.
- Extract features of all images one by one through HOG function.
- For each business, we will compute the mean feature vector among images that belong to it.
- Using 10-fold cross validation to get evaluate the results.

5.3 Evaluation For this project, we would like to build a restaurant photo classifier which is the multi-label classification problem. Therefore, Let D be a multilabel evaluation data set, L be a label, consisting of $|D|$, multilabel example(x_i,Y_i), i=1...$|D|$,$Y_i \subseteq L$. Let H be a multi-label classifier and $Z_i = H(x_i)$ be the set of labels predicted by H for example x_i.

The following metrics are used for the evaluation of H on D:

$$Precision(H, D) = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{Y_i \cap Z_i}{Z_i}$$

$$Recall(H, D) = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{Y_i \cap Z_i}{Y_i}$$

The F1 score can be interpreted as a weighted average of the precision and recall, where a F1 score reaches its best value at 1 and worst at 0. The traditional F-measure or balanced F-score is the harmonic mean of precision and recall:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

The F1 metric weights recall and precision equally, and a good retrieval algorithm will maximize both precision and recall simultaneously. Thus, moderately good performance on both will be favored over extremely good performance on one and poor performance on the other.

6 Conclusion

References and Notes

1. Foody, Giles M., and Ajay Mathur. *A relative evaluation of multiclass image classification by support vector machines* Geoscience and Remote Sensing, IEEE Transactions on 42.6 (2004): 1335-1343.

2. Wei, Yunchao, et al. *CNN: Single-label to multi-label* . arXiv preprint arXiv:1406.5726 (2014).

3. Lin, Yuanqing, et al. *Large-scale image classification: fast feature extraction and svm training* . Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. IEEE, 2011.

4. S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. *cuDNN: Efficient primitives for deep learning* . arXiv preprint arXiv:1410.0759, 2014.

5. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrel. *Caffe: Convolutional architecture for fast feature embedding* . arXiv preprint arXiv:1408.5093, 2014.

6. A. Krizhevsky, I. Sutskever, and G. Hinton. *ImageNet classification with deep convolutional neural networks* . In NIPS, 2012.

7. R. Girshick, J. Donahue, T. Darrell, and J. Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation* . In CVPR, 2014.

8. Dalal, N. and Triggs, B. *Histograms of Oriented Gradients for Human Detection* . IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005, San Diego, CA, USA.

9. scikitimage *http://scikitimage.org/docs/dev/auto_examples/plot_hog.html*.