Action Recognition Baseline for Large Scale Action Recognition

Jiaxuan Wang
University of Michigan
500 South State Street, Ann Arbor, MI 48109
jiaxuan@umich.edu

Introduction

Human activity recognition has been laboriously studied, but has always been a challenge in computer vision. The objective is to have computers "understand" human actions in video clips and/or in photos. Successfully solving this problem would translate to breakthrough in robotic vision, video surveillance, human computer interaction, multimedia retrieval, and so on.

The challenge in action recognition is mainly two folds. First, we are forced to learn millions of general action concepts with small datasets. Second, action recognition is a combination of multiple research topics, each challenging in its own way. I will briefly touch on these problems in the following paragraphs.

Although plentiful attempts have been made for tackling action recognition [3], none of them used a significant dataset. The typical setting for the state of the art action recognition is to use a dataset (eg. PASCAL VOC [4]) containing a limited number of activity classes (usually less than 20), and for each class (eg. Walking), only hundreds of images are used for training and testing. An example of such a dataset is people-playing-musical-instrument (PPMI) [5], which has a total of 7 classes, each having 100 testing and training images. Comparing the number of activity classes used in papers to the actual number of human activity classes (which one can approximate by enumerating all possible verb noun pairs), it should be clear that there's a huge gap between theory and practice.

Besides current datasets being suboptimal, current techniques in action recognition is a combination of methods from multiple research areas including object detection, human detection, body pose estimation, and scene context detection. For example, to determine whether a person is playing an instrument or not, it is not sufficient to simply train 2 binary classifiers that detect the presence of both instruments and people because we can still get false positives when a person is holding an instrument but not playing it. A key observation is to use image patches where the person of interest and the instrument interact together (eg. a small image region where the chin of a person and the violin touches). The question then becomes how to find such discriminative image patches automatically.

This project is part of a bigger effort that aims at addressing the issue of small size by building a large dataset containing hundreds of activity classes, each having thousands of images. The dataset is built by crawling the web and using data mining techniques to infer about the label for each retrieved image. In addition to automatic labeling, the resulting class is validated by people through the use of Amazon Mechanical Turk (AMT). My part is unrelated to dataset building, but to provide a baseline result for this newly constructed dataset. In essence, I implemented an algorithm in [1] (referred to as Bangpeng's method in the remaining part) to classify images in this new dataset.

Approach

As discussed in introduction, a key for recognizing action in still images is to find discriminative image patches. There are three questions we need to answer. The first is from what space we choose image patches; the second is what criteria is used for defining discrimination; and the third one is how we can efficiently search through the candidate set. Bangpeng's method covers all these issues.

The answer to the first question is straightforward. To fully capture the image space (referred to as deep sampling space in [1]), we consider every possible image patches that vary in center, height, and width. The second question is easy to answer as well, we define discrimination according to information gain (how less uncertain we become after splitting using the image patch feature). The third question is trickier, it should be noted that it is infeasible to do a vanilla grid search through the deep sampling space as the number of image patches grow exponentially with the number of pixels. Instead, we randomly sample image patches and appeal to the law of large number to ensure the coverage of the whole space.

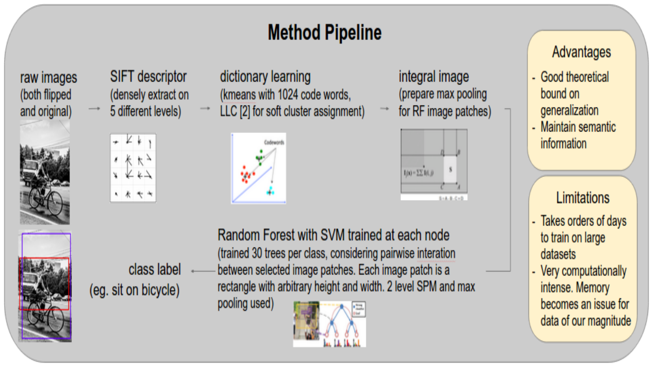
Before we jump into the specifics of Bangpeng's method, it is helpful to review the general setup for action recognition. Action recognition is usually formulated as a image classification problem. We want computers to automatically detect patterns in raw pixels and to categorize pictures accordingly. It can be applied to simple tasks like distinguishing a dog from a cat, detecting human faces and so on. A common ingredient in solving such a problem is machine learning, or more specifically, supervised machine learning (unsupervised algorithms are used but usually for the purpose of data cleaning like dimensionality reduction). First, labeled examples are used to train a mathematical model (a representation of the problem with parameters to "learn"), and then this model is used to predict the tag of unlabeled data (in this case action classes of images).

The key machine learning technique used in Bangpeng's method is Random Forest. Random Forest is essentially a combination of individually trained decision trees. By training, I mean learning the criterion to split the images into different tree nodes. The leaf node of each tree is a distribution over the classes of possible tags (based on the proportion of training examples on each leaf node). In predicting the tag of an image, we pass the image into every tree and cascade it down to the leaf nodes, and then we take the majority vote of trees to determine the most likely tag.

The key for Bangpeng's method lies in two terms 'discriminative' and 'random'. Unlike conventional Random Forest where at each node we split on only one feature, each tree node is a strong classifier that focuses on one or multiple image patches (leads to very low generalization error). Here, the algorithm for training a strong classifier is linear Support Vector Machine (SVM). A SVM is a maximum margin classifier and this is what makes Bangpeng's method 'discriminative'. 'Randomization' comes from the way I sample images and the way Random Forest learns the splitting criterion (both detailed in pipeline overview).

[6] justifies the theoretical validity of Bangpeng's method. The basic idea is that the less correlation between trees and the stronger each tree node classifier is, the better the generalization for Random Forest. In this case, random sampling from the dense sampling space ensures small correlation between trees, and linear SVM makes each node a strong classifier.

The original paper contains experiments made on three datasets. The result was promising that it was better in mean average precision than the winning algorithms of PASCAL VOC 2010 [4] in 7 out of 9 categories. Yao also used PPMI [5] and Caltech-UCSD Birds 200 (CUB-200) [7] as datasets to illustrate his method. As stated before, all of these datasets are trivial compared to the amount of images on the web so that it is worthwhile to benchmark its performance on this new dataset, HICO.


Method Pipeline
raw images (both flipped and original)
SIFT descriptor (densely extract on 5 different levels)
dictionary learning (kmeans with 1024 code words, LLC [2] for soft cluster assignment)
integral image (prepare max pooling for RF image patches)
Codewords
Random Forest with SVM trained at each node (trained 30 trees per class, considering pairwise interation between selected image patches. Each image patch is a rectangle with arbitrary height and width. 2 level SPM and max pooling used)
class label (eg. sit on bicycle)
Advantages
- Good theoretical bound on generalization
- Maintain semantic information
Limitations
- Takes orders of days to train on large datasets
- Very computationally intense. Memory becomes an issue for data of our magnitude

Graph 1 Overview of Bangpeng's pipeline

The pipeline of Bangpeng's methods is summarized in Graph 1 and detailed in the following paragraphs. Here the exact setting is based on Bangpeng's original paper (using PASCAL VOC 2010).

a) Data preprocessing:

- *Foreground image extraction:*
Foreground images are those images that are within the bounding boxes of each background image (the input pictures). They are extracted in this step.
- *Training file boosting:*
The number of training examples are doubled by including flipped images.
- *Image resizing:*
To make computation feasible for the next few steps, all foreground images are resized to an area within 300 by 300 pixel region.
- *SIFT feature extraction:*
SIFT features are densely extracted for both foreground and background images with grid spacing of 4 pixel, sigma edge 0.8, and patch sizes of 8, 12, 16, 24, and 30 for 5 different settings
- *Dictionary learning:*
Out of all the images, 4000 are randomly sampled, and for each image selected, one patch size of SIFT is chosen, and a maximum of 4 million SIFT features are used. Kmeans is used to reduce dimension with a code word size 1024. Then Locality-Constrained Linear Coding (LLC) [2] is applied with the learned dictionary to softly assign SIFT features to 1024 clusters.
- *Preserving locality information of images:*
For background images, only 4 locations of LLC features are preserved, while for foreground images, a SIFT grid space of 8 and step size 4 is preserved.
- *Integral image*:
For fast computation of max pooling, integral images are computed for each training/test examples on every SIFT locations using dynamic programming.

b) Random Forest training

- *hyperparameters setting:*

A forest with 100 trees are trained with maximum tree depth of 10. Each leaf node contains at least 40 training examples and nodes are split according to information gain.
- *Feature selection:*
For each image, the feature space is the set of all possible windows within the picture. Practically, it is done by drawing 4 uniform(0,1) random variables as the top left and bottom down position of a rectangular window and resize the numbers to fit the image. A minimum region size is set to be at least 40% of the image width and height. Moreover, to incorporate global information, the 2 by 2 background image LLC feature is concatenated to the selected feature, and these features are max pooled to store only relevant information. Also note that, the feature selected in this step may not be a single image patch. It can also be the interaction of 2 image regions by applying standard histogram operations like intersection and summation on the 2 features (the reason being that for action recognition, the connection between an object and a subject is important).
- *Number of features to split on each node:*
One assumption most ensemble learning methods make is that each learner is independent from each other so that the majority vote are probabilistically in favor of such methods. This hold true for Random Forest. Here, 25 and 50 features (image regions) are chosen for splitting the root node and first level nodes respectively, while 100 features are used to split all other nodes (so as to maintain independence of trees).
- *Linear SVM splitting:*
The actual splitting of nodes is based on training an SVM with a linear kernel. Here, a binary label is randomly assigned to action classes for training convenience. It has been shown in [6] that Random Forest generalizes better by combining strong classifiers like SVM.

c) Evaluation of result

As this is a classification problem, the most natural metric to use is accuracy. However, for the 9 classes of VOC action challenge, the splitting of positive and negative examples are not balanced so that accuracy itself means nothing. The metric used in this paper is mAP (mean average precision). The idea is to compute average precision (sweeping precision at each recall point and average the precision) for each action class and take the mean across classes. This metric takes into account the ranking of probabilities that many other metrics fails to account for.

Implementation

The implementation for this project is hard. There are 2 levels of difficulties I overcame.

First, the original implementation is buggy and I had to dig through every piece of code (68 poorly documented files not counting libraries and VOC development code) to track down problems. As the code was written as a mix of C++ and matlab, I don't get the luxury of using graphical interface to debug as segfault instantly crashes it. As I begin to appreciate the complexity of the project, I found that the file structure required by the code was very stringent and the author made serious assumptions that may not hold. For example, he never documented the location of foreground testing files (not the PASCAL test file format, but a format that he creates with ground truth and filenames, and this file cannot be found anywhere on the web) and the way to generate those files, I recreated those file after speculating the usage of them in the code. Moreover, even his code in C++ was flawed. Not only did he hard coded some of the parameters (such as code size of 1024) that was intended to change in the matlab file, config.m, he also made incorrect assumptions in testing random forest by mixing 2 variables of the same name in different scopes together (check out his code in dttest.cpp for the variable classNum and you will find that he used the variable inconsistently across functions, creating debugging nightmare). It took me an entire month with 2 days per week just to decrypt and recomment the 68 (12 cpp and 56 matlab) files (his coding style is bad, copy pasting from all places, and comments are irrelevant).

Second, the concept is difficult to grab for beginners. There are too many lingo like Random Forest, SVM, kernel, LLC, dictionary learning, Spatial Pyramid pooling, bag of visual words that I don't understand initially. Even SIFT extraction is different from the taught key point extraction but dense extraction. This difficulty is fun to overcome as I learned a lot by inspecting the code and search through the Internet. One thing notable is that Bangpeng wrote his own code for Random Forest and SVM learning in C++ and I debugged from ground up those code and actually caught a bug in the way he read in existing trees.

The code for Bangpeng's method was provided on his website (http://ai.stanford.edu/~bangpeng/) but is no longer available as he deleted everything on his personal page. Components of his code includes bounding box image extraction, SIFT extraction, Kmeans, LLC extraction, Random Forest with linear SVM training, and Random Forest testing (output the result of each tree for test images). The following is a time breakdown of running the code on PASCAL VOC 2010. Note that both flipped and non flipped images are used so that the amount of work is doubled.

Total time:
original setting: 10 hr, parallelized: 5 hr
Configuration:
16 core
Parallel code time:
1 hr 20 min, increased to 1 hr 41 min
Non parallel code time:
8 hr 40 min, decreased to 3 hr 10 min
** **Background:** 3 hr 20 min
- *Generate input file:*
0.206171 s

- *SIFT feature extraction:*
3429.139251 s (1 hr, (228+226)*2 = 908 images, parallel, so time per core per image is 60.42 s)
- *Generate Dictionary:*
7479.134036 s (2 hr, not parallel, 1 million features, constant time)
- *LLC pooling:*
867.695421 s (15 min, parallel)
- *Combining LLC:*
43.847333 s (parallel, without parallel: 688 s = 12 min)
** **Foreground:** 1 hr 10 min
- *Create Foreground Image:*
27.734619 s (301+307 = 608 images)
- *generate input file:*
0.497327 s
- *SIFT feature extraction:*
1222.651143 s (20 min, (301 + 307) * 2 = 608 * 2 = 1216 images, parallel, so time per core per image is 16.88 s)
- *Generate Dictionary:*
2101.546122 s (35 min, not parallel, 0.26*10^6 features, constant time)
- *LLC pooling:*
324.397295 s (5 min, parallel)
- *Combining LLC:*
216.402459 s (parallel: 4 min, without parallel: 1 hr)
** **Other:** 5 hr 30 min, parallelized: 21 min
- *Ids generation:*
0.108695 s
- *Output binary feature:*
0.552356 s
- *Train RF:*
19834.07494 s (5.5 hr, 100 trees, 198 s = 3.3 min per tree, if parallel, need 21 min)

The original code only parallelizes SIFT and LLC extraction, however, as noted in the time breakdown, the time for completing the whole pipeline can be halved if we also parallelized Random Forest training. The idea is to train each tree independently from each other on different processors and combine the majority votes manually. The original code structure makes it hard to do so as trees are trained in one pass and saved in the same directory. I ended up rewriting the reading and outputting tree functions to suit my need. Also, for some reasons, a portion of files are corrupted when generated through each step (possibly due to interruption in submitting Flux pbs files). I also wrote helper functions to check whether files are corrupted based on the expected output of each file (for example, to check whether a tree is valid, I know that for binary classification tasks, the number of classes recorded at each node must be 2, and check accordingly), and then regenerate those corrupted files.

Besides issues with getting the original code to work in a reasonable amount of time, the real challenge comes from adapting the code for the new dataset. Note that the time breakdown is based on configuration for **16** cores and on a dataset restricted to **9** action classes with **454** images total. For the HICO dataset we collected, we have **600** action classes with a total of **45786** images, which is much larger. This means that to run on our dataset, Bangpeng's method would take at least a month. Time is not the only concern, memory is also an issue. Some steps such as Random Forest training/testing and output mat files (write to disk the integral image features Random Forest training would need) takes 9 GB for each class because of the large number of images it has to read in (5000 images for training, 9000 for testing). Since each node on Flux only has 4 GB RAM, it is far from feasible to running (one can use large memory allocation but at the cost of high expense). The solution is to a) use more computational resource b) further parallelize the code c) optimize for our problem (exploit information that the original problem does not assume, for example, we treat each class as a binary classification problem instead of as a multi-class classification problem). I ended up using **120** cpus, manually split **600** action classes into **100** groups and compute for each group in parallel (the reason for not just parallelize for 120 cpu, for example using parfor for 600 classes, is that the actual speed is 3 times faster if I manually split those classes. This phenomenon can be explained by locality of resources so that the time to complete a task is linearly related to the number of cpus used). To reduce memory usage, I'm not using flipped images anymore because the amount of training images is large enough. I also lowered the number of images sampled for dictionary learning from 4000 to 100 images (I don't expect it to make much a difference according to the law of large number). I also rewrote the output mat file function to process one image at a time instead of doing them in a batch. Also, the number of trees trained for each action class is lowered from 100 to 30 (expected not to lower the performance as the original problem has 10 action classes, but for this new problem, each class is a binary classification problem). For fast computation, I wrote code to cache the result for Random Forest testing, integral image (originally a single mat file is created for each action class, but since many classes share the same images, it is more computationally efficient to cache individual images), SIFT extraction, and LLC extraction (although the last two are already cached in the original setting, they do not output to a combined directory so that many computation are wasted on already computed image patches. I rewrote a function to redirect the output). Also, note that the HICO dataset doesn't have bounding box information so that the entire process for handling background images is eliminated.

Experiments

I did three experiments on the dataset. The first one replicates the result in [1], the second one produces result on HICO, and the third one produces result using a subset of HICO that has higher quality of annotation.

a) Replication of paper result

Action Classes / AP (%)	Phoning	Playing Instrument	Reading	Riding bike	Riding horse	Running	Taking photo	Using computer	Walking	Overall
Reported in the paper	45.0	57.4	41.5	81.8	90.5	89.5	37.9	65.0	72.7	64.6
Replicated result	42.663	44.677	37.207	75.066	89.279	79.445	26.907	38.752	73.207	56.356

Table 1 Comparison of Performance evaluated using mean average precision

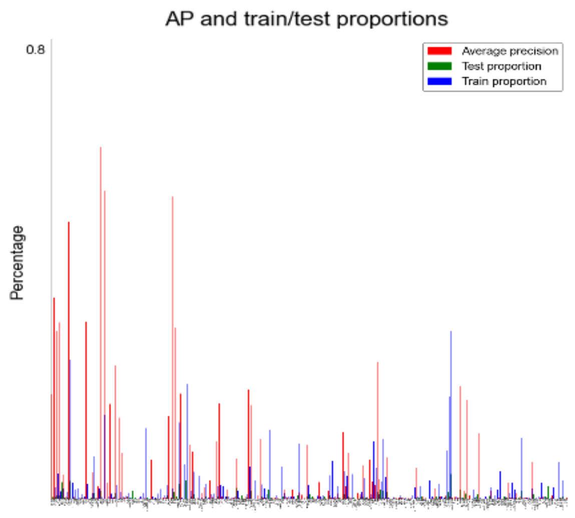
To make sure that I successfully debugged Bangpeng's code, I compared my result on PASCAL VOC 2010 to the result reported in his paper.

Table 1 indicates that my modified implementation on Bangpeng's method achieves comparable result from that of the original paper (the trend in 9 classes are similar). Possible explanations for differences in performance include:
i) The replication of result only uses training images (listed in train.txt, containing 226 images) for training, but the original paper uses both training and validation set (listed in trainval.txt, containing 226+228=452 images) for training, which doubles the amount of training images (note that I cannot replicate the paper result using both train and validation set because the label of test set is not publicly available, so I used the validation set to report mAP. It is dubious as to where the original author get the test labels).
ii) The original paper sampled 4000 images for SIFT extraction, but I only sampled 100 images (due to space limit because even 100 images produces 1 million SIFT for dictionary learning).

b) Bangpeng's method applied to HICO

After verifying my implementation, I applied Bangpeng's method on our HICO dataset. The result is summarized in the following graph.

Here, the red bar represents the average precision for an action class, the green bar is the the proportion of positive test images, while the blue bar is the proportion of positive training images in a class. As one can verify, the average precision for each class is much higher than random guesses based on class proportion.



method	mAP (%)
B. Yao's Method	**4.9116**
FV + SVM	4.25
DNN feature (ImageNet) + SVM	18.02
DNN feature (fine-tune) + SVM	18.17
HOCNN feature + SVM	7.12

Table 2 comparison of Bangpeng's method with other baselines

Compared to other baselines done in our research group, Bangpeng's method works reasonably well. It is comparable to the fisher vector and SVM combination but far behind deep learning methods. This is expected as deep learning is a super star in machine learning and it is still under active research.

c) Bangpeng's method applied to refined HICO

The vanilla HICO dataset contains incorrect annotations so that there are noises in the annotation itself. In this experiment, I only used a subset of HICO where the annotations are known to be correct.

The mean average precision in this setting is calculated to be **38%**, which is far higher than the performance on the original dataset. As I manually verified the action class *sit on bench*, I found more than 50 images to be false negatives out of total 5000 training images. This again shows the importance of good annotation.

Graph 2 is a snapshot of the resulting discriminative image patches. I basically sampled one image from each class and draws the bounding boxes using the image patches learned in one particular tree for each class. It should be noted that the patch selected by the Random Forest matches human intuition by focusing on positions where a person and an object interact.

Graph 2 HICO dataset with discriminative image patches

Conclusion and Future work

Bangpeng's method is a winning method for PASCAL VOC 2010 and 2011. In this project, I show that it performs reasonably well on the HICO dataset. It has the following advantages:
a) Unlike conventional decision trees, the algorithm uses a strong classifier (linear SVM) at each node and combines information at different depths of the tree to effectively mine a very dense sampling space.
b) Bangpeng's method not only increases tree strength, but also maintains low correlations between the trees thus achieving low generalization error.
c) Banpeng's method provides semantically meaningful information (producing discriminative image patches that resemble what a human being would look at).

The drawback of Bangpeng's method lies in the huge amount of work to be done in both training and testing. Even after I parallelized and optimized the code, it still takes six days to complete the whole pipeline using 120 cpus, which makes it far less suitable for real use.

Future researchers should focus on cleaning the HICO dataset more as my third experiment shows better results using a cleaned version. Also, I would advice future researchers to approach action recognition from a deep learning perspective instead of using Bangpeng's method as deep learning is far more promising.

References

[1] B. Yao, A. Khosla, L. Fei-Fei, Combining Randomization and Discrimination for Fine-Grained Image Categorization

[2] Locality-constrained Linear Coding for Image Classification, Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas Huang, Yihong Gong.

[3] Daniel Weinland, Remi Ronfard, Edmond Boyer, A Survey of Vision-Based Methods for Action Representation, Segmentation and Recognition, 2010.

[4] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results.

[5] B. Yao and L. Fei-Fei. Grouplet: A structured image representation for recognizing human and object interactions. In CVPR, 2010.

[6] L. Breiman. Random forests. Mach. Learn., 45:5–32, 2001.

[7] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD birds 200. Technical Report CNS-TR-201, Caltech, 2010.