Final Report:

Evaluation of Locality-constrained Linear Coding for Image Classification

Liu, Neil
Jiang, Ruochen
Xie, Wenqiao

Abstract

The purpose behind image classification is a very simple one, allowing computers to be able to recognize objects with high accuracy. Because of a lack of powerful computer hardware, the solution to this simple problem however has eluded researchers despite most of the theory already being present. To put this into perspective, our simple dataset contained roughly 12 gigabytes of information and took 3 hours to compile on high end processors and for more practical datasets storing the data can be well in the 100’s of terabytes. This hardware limitation is perhaps one the biggest reasons why this image classification technology has improved so substantially in recent years. Such hardware upgrades have encouraged waves of innovation from facial recognition to the creation of potentially self driving cars, allowing for improvements in both security, through the ability to create more substantial counterterrorism methodologies, and transportation, through the possible elimination of traffic and accidents. Because image recognition can potentially produce so many technologies that can substantially benefit society, we decided to study this very broad and fascinating topic. We specifically chose to focus our efforts on the LLC (Locality-constrained Linear Coding) encoding method because it dealt with concepts we felt we had a grasp on.

Keywords: feature extraction, SPM, LLC, image classification, Caltech 101.

I. INTRODUCTION

To better understand the LLC encoding algorithm, we first have to approach image classification from a much higher level. In order to describe each step in a more understandable way, we like to use an analogy comparing an image to a passage of words from a new language. The first step in successfully finding this passage in your own language is to extract important words from this passage. The second step would be match these words with a known dictionary to discover what words these actually are in your own language. Finally, you count how many of each word is in the passage, this method can be modified for accuracy by also considering where those words occur, and
then use this information to compare with known passages to classify which passage this is.

An image is classified in a very similar way and generally can be broken down into 3 major steps: feature extraction, encoding/pooling and finally classification. Instead of words being extracted from a passage, features are extracted from an image using one of two methods, SIFT (scale invariant feature transform) or HOGS (histogram of oriented gradients). Both of these methods use the general idea of transforming unique pixel collections into a unique vector and thus transforming an image into mathematically workable matrix. The next step is to compare each feature with a dictionary of known features, also called the codebook, like how one would compare words from an unknown language to translations in a dictionary. In order to generate this codebook, one takes all extracted features and tries to condense them into n features (the community normally uses 1024, 2048 or greater) through the minimization of proximity error and can be done through the K-means method. This essentially translates unknown features into known features which then can be compared and analyzed. The above step is known as encoding and has 3 major algorithms, VQ (Vector Quantization), ScSPM (Linear Spatial Pyramid Matching using Sparse Coding[SC]), or the method we focused our research on, LLC. Finally, the number of features are then counted and normalized into one vector, to improve accuracy like in the analogous case with the passage of words, one can can count features in specified areas of the image using a method called SPM (Spatial Pyramid Matching). These vectors are then used to classify the images using classification techniques such as SVM (support vector machine), which essentially compares how many features each image has with other images to match images together.

II. ENCODING METHODS

To understand the motivation behind the algorithm we studied, it is crucial to first look at the past algorithms.

Vector Quantization [1]:

The first algorithm is the VQ algorithm which uses this optimization equation:

$$\arg \min_{c} \sum_{i=1}^{N} ||x_i - Bc_i||^2$$

$s.t. ||c_i||_F = 1, ||c_i||_I = 1, c_i \geq 0, \forall i$

The problem with this method is that the condition is very limiting and requires each code to have only one non-zero value and that non-zero value has to be one. Using this methodology, one would have to be very certain that the feature in the dictionary is corresponding to the feature extracted from the image, if not then there would be large accuracy errors, which were present in VQ tests. The benefit of the VQ algorithm is that it is easy to solve due to the constricting constraint.

ScSPM [1]:

Secondly, the ScSPM algorithm is a sparse coding algorithm designed to use
sparse parameter to create its encoding equation:

$$\arg \min_C \sum_{i=1}^{N} \|x_i - Bc_i\|^2 + \lambda \|c_i\|$$

However, the major issue with this algorithm is that it focuses on creating codes that are sparse rather than codes that may represent the data well. Essentially instead of matching extracted features with features in the codebook that mostly resemble the extracted feature, the ScSPM method matches extracted features with features in the codebook to keep the code matrix sparse. This can be seen almost as a way of saving time while sacrificing slight accuracy. With these two algorithms there seems to be a tradeoff between the accuracy and the runtime.

LLC [1]:

The authors of the LLC encoding paper tried to tackle this issue by creating an algorithm that was both efficient and accurate. By changing the optimization equation to:

$$\min_C \sum_{i=1}^{N} \|x_i - Bc_i\|^2 + \lambda \|d_i \otimes c_i\|^2$$

$$s.t. 1^T c_i = 1, \forall i$$

Where: $$d_i = \exp \left( \frac{\text{dist}(x_i, B)}{\sigma} \right)$$

And: $$\text{dist}(x_i, B) = [\text{dist}(x_i, b_1), ..., \text{dist}(x_i, b_M)]^T$$

Thus this algorithm can speed up runtime. In fact, the LLC encoding algorithm is able to achieve .3 second recognition time while showing higher accuracy than previous methods. Therefore, the authors were able to create an algorithm that essentially solved the speed vs. accuracy conundrum.

III. RESULTS

In our experiment, we used the Caltech 101 as our dataset and we used the codebook provided by the authors of the paper. For the pooling step, we used max pooling with normalization; for the classification step, we used one vs all linear SVM. By doing so, we try to mimic the experimental parameters presented in the paper, so we can compare our result to that in the paper. Here, we compare the results in two ways.
In **Table 1** and **Figure 1**, given different numbers of training images, we compute the prediction accuracy for each class, and then take the simple average of these accuracies. This method of calculating prediction accuracy is the same as the method utilized by the paper.

<table>
<thead>
<tr>
<th>Number of Training Images</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper Results</td>
<td>51.15%</td>
<td>59.77%</td>
<td>65.43%</td>
<td>67.74%</td>
<td>70.16%</td>
<td>73.44%</td>
</tr>
<tr>
<td>Our Results</td>
<td>47.63%</td>
<td>57.78%</td>
<td>63.37%</td>
<td>66.93%</td>
<td>68.92%</td>
<td>70.67%</td>
</tr>
<tr>
<td>Deviation</td>
<td>3.52%</td>
<td>1.99%</td>
<td>2.06%</td>
<td>0.81%</td>
<td>1.24%</td>
<td>2.77%</td>
</tr>
</tbody>
</table>

**[Table 1]**

We were able to perfectly predict 7 classes (**Table 2**) while the paper was able to perfectly predict 13 classes. Here are some possible reasons for the discrepancy between our result and the result in the paper. The primary reason is that we are using a different codebook from that in the paper, we are using a codebook with 1024 bases while in the paper, they used a codebook with 2048 bases. Also, using different methods of feature extraction (we use SIFT, while the paper uses HOG) and different SVM parameters in the classification step can contribute to the difference of result in some degree.

**[Table 2]**

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Average Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_side</td>
<td>100%</td>
</tr>
<tr>
<td>inline_skate</td>
<td>100%</td>
</tr>
<tr>
<td>metronome</td>
<td>100%</td>
</tr>
<tr>
<td>minaret</td>
<td>100%</td>
</tr>
<tr>
<td>pagoda</td>
<td>100%</td>
</tr>
<tr>
<td>scissors</td>
<td>100%</td>
</tr>
<tr>
<td>trilobite</td>
<td>100%</td>
</tr>
</tbody>
</table>

In **Figure 2**, we took the prediction accuracy for each testing image and then took the simple average of all these prediction accuracies by adding them together and dividing the sum by the total number of testing images.

**[Figure 2]**

As we can see the results, the second way of calculating accuracy is much higher than that of the first way. This phenomenon can be explained through weighted averages. When we take the average across all images, we are essentially weighting each image equally; however, when we take the average across all classes, we are weighting each image differently depending on which class it is apart of and how many images are tested in that class. Therefore, if images in large classes have higher accuracies than small classes, then the
equal weighting method would be higher than the weight by class as each class is weighted equally and thus more images means a lower weight.

IV. DISCUSSION

In order to improve the prediction in the future, we suggest some methods in this case.

1. Using More Partitions:
   Currently we are using a 3 levels pyramid: [1,2,4]. Increasing the amount of levels of the pyramid should improve accuracy, for example [1,2,4,8]; however, this would be more computationally intensive. Using more partitions for each pyramid layer should also improve accuracy, for example [1,3,9]; however, doing this may also be computationally expensive.

2. Using a Larger Codebook:
   Currently we are using a codebook with 1024 bases. If we use a codebook with more bases (like 2048), our results should improve.

3. Using an Updating Codebook
   Currently we are using a non-updating codebook. In the paper, the authors try to update their codebook by using an algorithm called incremental codebook optimization. However, this method does not show significant improvements in accuracy. There in fact is only a .4-1.3% improvement when using an updating codebook produced by the algorithm above versus using a fixed codebook. Because of this insignificant change in accuracy, we decided to not use this codebook optimization method [1]. In this case, learning bases using the Lagrange dual [2] would be a good approach, which is present in the sparse coding paper.

V. LESSONS

We chose to not do experiments on any other datasets because of time limitations and computing hardware constraints. Because the Caltech 101 dataset took over 3 hours to extract features and pool, we expected the other datasets to take much more time and potentially force us to purchase hardware to store. For these reasons, we decided to only work with the Caltech 101 dataset.

Before we classify our pooling results, we wanted to save these pooling results into files with labels first. Therefore, we learned how to use the Python package, Pickle, which allowed us to be able to not only save our pooling results but also be able run the encoding pooling algorithms on multiple computers, saving us valuable time. One of the major issues we encountered is a structure mismatch between Matlab and Python. Struct in Matlab has to be converted to a dictionary in Python which is then broken up into lists and arrays and therefore, we learned how to use lists and Numpy arrays to represent original matrices.

We learned that because SVM involves gradient descent method, the more iterations the higher the accuracy. Therefore, even by training the SVM model with a dataset appended with itself, there seems to be improvements in
accuracy, but it sacrifices runtime efficiency. Moreover, we also need to be aware of choosing a reasonable value of step size for this optimization method, otherwise, it will probably diverge.

Last but not least, as we had to convert Matlab code into Python, we learned about Python coding and the limitation of the Numpy package. Some of the native packages in Matlab are not available in Python and thus we had to discover methodologies of recreating that function in a new programming language.

VI. REFERENCE
