Final Report: Object Recognition Using Large Datasets

Ashwin Deshpande

12/13/07

Object recognition is a difficult problem due to the large feature space and the complexity of feature dependencies. First, there exist positional complexities resulting from the 3D position and orientation of the object as well as the 3D position and orientation of the camera. Further, changes in lighting, background, and occlusion can create dramatically different images for the same object. In addition, to these complexities, we try to perform object recognition over classes of objects (mugs, pliers, scissors, etc.) which contain variances between objects in the same class.

While there has been a steady trend in representing object characteristics with increasingly complex models and features, we instead chose a different approach by artificially generating a large training set and applying simple algorithms on the result. This method has several advantages and disadvantages. On the positive side, using more data almost always yields better results. On the other hand, the generated training data may not be an accurate representation of reality and may create an artificial bias. Furthermore, when dealing with millions of images simultaneously, special precautions must be taken to respect the strict hardware constraints.

This paper first discusses the image generation process. Next, it explores two nearest neighbor related algorithms. The first uses cover trees, and the second implements modified Nister trees. Finally, the paper ends with future work and conclusions.

Image Generation

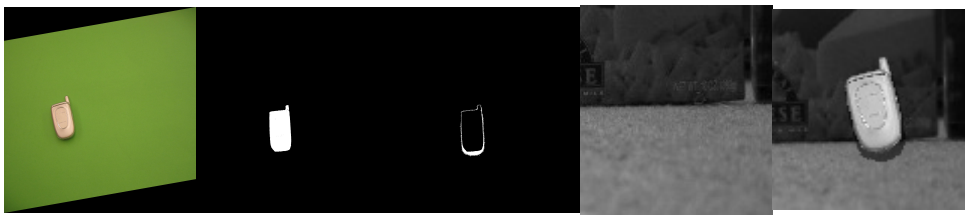

Figure 1: Parts of the image generation process. From left to right: green screen image of object, object mask, shadow mask, background, combined image

The images were generated in a very similar manner to a previous paper [1]. We used an existing set of about 1400 photographs and regions of interest of objects (150 per class) with a green screen backdrop and a stock collection of about 1200 office backgrounds. Masks for each object and its shadow were extracted using intensity thresholds and smoothing (see Fig. 1 for an example). To generate an image, both the background and object were subjected to perspective changes via

affine transformations in the homogenous coordinate space including stretching, translations, and rotations. Next, the object was copied onto a randomly selected region of the background. Finally, a random shadow intensity was projected onto the background. Using this method, millions images of sizes 50x50 to 500x500 were created.

Cover Trees

We first experimented with cover trees to perform nearest neighbor calculation. A cover tree is an efficient data structure which allows for quick nearest neighbors calculations by hierarchically organizing the data into a tree with guarantees on both the construction time and query time. As we performed nearest neighbors calculations on the order of million images, the speedup was both large and computationally necessary.

The first step involved feature extraction. At first, 1 million 100x100 images were generated for each class. Features were set to be individual pixel values (0-255). As performing PCA on the dataset required the infeasible task of computing the eigenvectors of a 10000x10000 dimensional matrix, we eventually scaled back image size to 50x50. Via PCA, the number of dimensions were safely reduced to 100 (with the eigenvalues of the last component vectors being essentially zero).

Following this, these vectors were fed into a cover tree. The first test was object detection. This problem attempts to answer questions of the form: "Is there a flipphone in this image?" The training set was divided such that half was composed of positive training examples from a single class and the other half was a random assortment of images from the other classes as negative training examples. To test the data, a set of 1800 real test images (200 per class) were queried against each created tree (see Fig. 2). Performance was measured as the sum of proportions of true positives and true negatives weighted equally. In addition to performing 1-nearest neighbor and 10-nearest neighbor search, we also attempted to discover any consistent distributional bias between real images and generated images. To do this, we calculated an approximation of the average and covariance of both the training data and half of the testing data. Then, we calculated the difference of averages and axis-aligned variances between the two. By simple linear shifts, the other half of the testing data was subjected to these transformations.

Overall, increasing the training set size did seem to have a noticeable impact on performance. In addition, for telephones and hammers, bias shifting the testing data caused significant improvements in performance indicating a consistent bias of image artifacts in the generated images.

The second test for cover trees was object classification. This problem attempts to answer questions of the form: "What object is in this image?" In this case, the training set was divided equally among all classes. As we used 9 classes in these experiments, an accuracy of greater than 11% indicates learning. As can be seen in Fig. 3, again increasing the training set size did seem to have a noticeable impact on performance and as before some object classes like telephones enjoyed a significant accuracy boost by normalizing the testing data. It is also interesting that 1-nearest neighbor seems to always outperform 10-nearest neighbors.

Overall, cover trees did yield results showing that using larger datasets of generated images can improve performance in image detection and recognition. However, cover trees seem to be bound by an upper limit of training set size due to memory constraints as each the features for each training point must be remembered. Thus, this precludes the use of cover trees for tens of millions of generated images on commodity hardware.

Modified Nister Trees

The Nister tree is a vocabulary tree specifically designed for distance computations between images based upon localized image patches rather than entire images. In this model, each image can be decomposed into a set of "interesting" image patches. Then, the distance between a pair of images is some aggregate of the distance between every pair of image patches. While this approach can be formalized to clearly define the distance between a pair of images, it cannot be used in conjunction with more traditional nearest neighbor algorithms like the cover tree as the distance computations are too slow. Thus, as an alternative, the modified Nister tree can efficiently ignore distance computations between very different feature vectors via a greedy tree search and enable efficient indexing and querying of large image datasets.

The first step in using Nister trees is feature generation. Initially, we tried using 50x50 and 100x100 images; however, the features extracted from these small images were unreliable. Ultimately, we settled on generating 10000 images for each class of size 500x500. Each image was broken down into a set of about 100-500 features. At first, we used SIFT to generate features for the image patches. However, we switched to SURF features as SURF features frequently outperform SIFT in the literature and each SURF feature takes 144B instead of 512B or SIFT to describe an image patch comparably.

The next step was generating the Nister trees. This involved plotting all features (localized image patches) from all training images in feature space. Next, k-means was applied hierarchically to generate a tree of feature clusters. In our implementation, we chose k=10 as a reasonable balance between performance and quality [6]. In the literature, there were a set number of levels for k-means clustering. However, we allowed a variable number of levels and instead chose to terminate the hierarchical clustering if the number of features in the leaf cluster was less than a threshold or the entropy of cluster with respect to the number of features in each class was less than another threshold. The first condition stops the creation of a superfluous cluster layer, and the second eliminates the need for unnecessary hierarchical subdivision.

In building the Nister trees, the largest tree built involved using 10000 images from each of 9 different object classes. This results in 90000 total images, 28.8M features, or 3.9GB of input data. Thus, due to hardware constraints, we had to adopt an incremental approach to building the Nister tree. We built the the first level of a Nister tree using only a small fraction of the training data. This provided adequate clustering for the top of the tree. Then, 10 times the number of features were binned into the preset clustering of the previous round, and the second level of the Nister tree was generated. This procedure was repeated multiple times until all 28.8M features could finally be inserted into the tree, at which point the tree was allowed to grow unrestricted. This approach was taken as the effective problem size was kept relatively constant at each level while the number of problems increased exponentially. This meant that only a small percentage of the data would need to be loaded at any point and memory overflow problems were less likely to occur. In addition, as this approach limits the amount of data of each subproblem, k-means clustering can be quickly computed in parallel. As the Nister tree groups clusters of features together, the 3.9GB of input data can be compacted into 500MB.

The original Nister tree is defined to measure the distance between a pair of images to judge whether the images describe the same object. In our case, we are interested in matching a query image to an abstract class. Thus, we have to modify the scoring metric. For a query image Q, we define q_i to be the number of features of Q that pass through leaf node i on the Nister tree. Let $N_{i,x}$ be the number of images of class x stored under the node i. We define the score S between Q and class x to be:

$$S(Q,x) = \frac{1}{N_x}\sum_i \frac{q_i}{\log(\frac{N_i}{N_{i,x}})}$$

where

$$N_x = \sum_i N_{i,x} \qquad N_i = \sum_x N_{i,x}$$

This scoring metric is a combination of multiple heuristics. First, the score is normalized with respect to the number of features for each class. Next, each node is given weight proportional to its prevalence in the query image. Last, we reward nodes with low entropy with respect to the class in question.

We tested Nister trees of size up to 90000 images for object classification (see Fig. 4). Nister trees seem to perform much better than cover trees for objects with unique features such as telephones and watches. Initially, we believed that using Nister trees on generated data would not be effective as SIFT/SURF feature extraction is supposed to be invariant to the minor affine transformations performed in image generation. To test this, we used 150 green screen images of each class of object as a control. While in theory, the Nister tree using green screen images should have performed better as it only learned features which describe the objects rather than features from the backgrounds, it turned out that the performance of Nister trees does improve with large generated training sets.

Future Work and Conclusions

We are still experimenting with various ways in which to compress the size of Nister trees to be able to store a larger set of information. Increasing the subdivision thresholds would allow this, but may also reduce the quality of query results from the tree. Instead, we can tackle the main source of space usage, the storage of the centroids of clusters. When processing a particular node in a Nister tree, we can perform an outer loop of greedy feature selection with an inner loop of k-means clustering. This will yield only the most relevant dimensions for clustering. Thus, the "centroid" of a cluster can then be defined as an axis-aligned hyperplane with the exception of the few relevant feature dimensions. Furthermore, the distance between a hyperplane and a point can be calculated in time linear to the number of non-axis-aligned dimensions, so queries will be speeded up as well.

It is clear from the results presented in this paper that object detection and classification can be significantly improved by using large datasets of generated images. By using cover trees on the pixel values of images and modified Nister trees on localized image patches, accuracy generally improved with training set size. This prompts the question if object recognition will benefit from tens of millions or hundreds of millions of generated images. The only obstacle in answering that question lies in transforming the problem and representation for efficient use given memory and computation constraints.

(Datasets are not readily available as they currently consume about 600Gb of space. I collaborated with Ashutosh Saxena to discuss many of the ideas and methods presented in this paper.)

References

[1] Anonymous. A Fast Data Collection and Augmentation Procedure for Object Recognition.

[2] H. Bay, T Tuytelaars and L.V. Gool. SURF: Speeded Up Robust Features. *ECCV*, 2006.

[3] A. Beygelzimer, S. Kakade and J. Langford. Cover Trees for Nearest Neighbor. *ICML*, 2006.

[4] L. Fei-Fei, R. Fergus and P. Perona. Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories. *CVPR*, 2004.

[5] Lowe, David G. Object Recognition from Local Scale-Invariant Features. *ICCV*, 1999.

[6] D. Nister and H. Stewenius. Scalable Recognition with a Vocabulary Tree. *CVPR*, 2006.

[7] T. Serre, L. Wolf and T. Poggio. Object Recognition with Features Inspired by Visual Cortex. *CVPR*, 2005.

[8] J. Sivic and A. Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. *ICCV*, 2003.

Appendices: Graphs

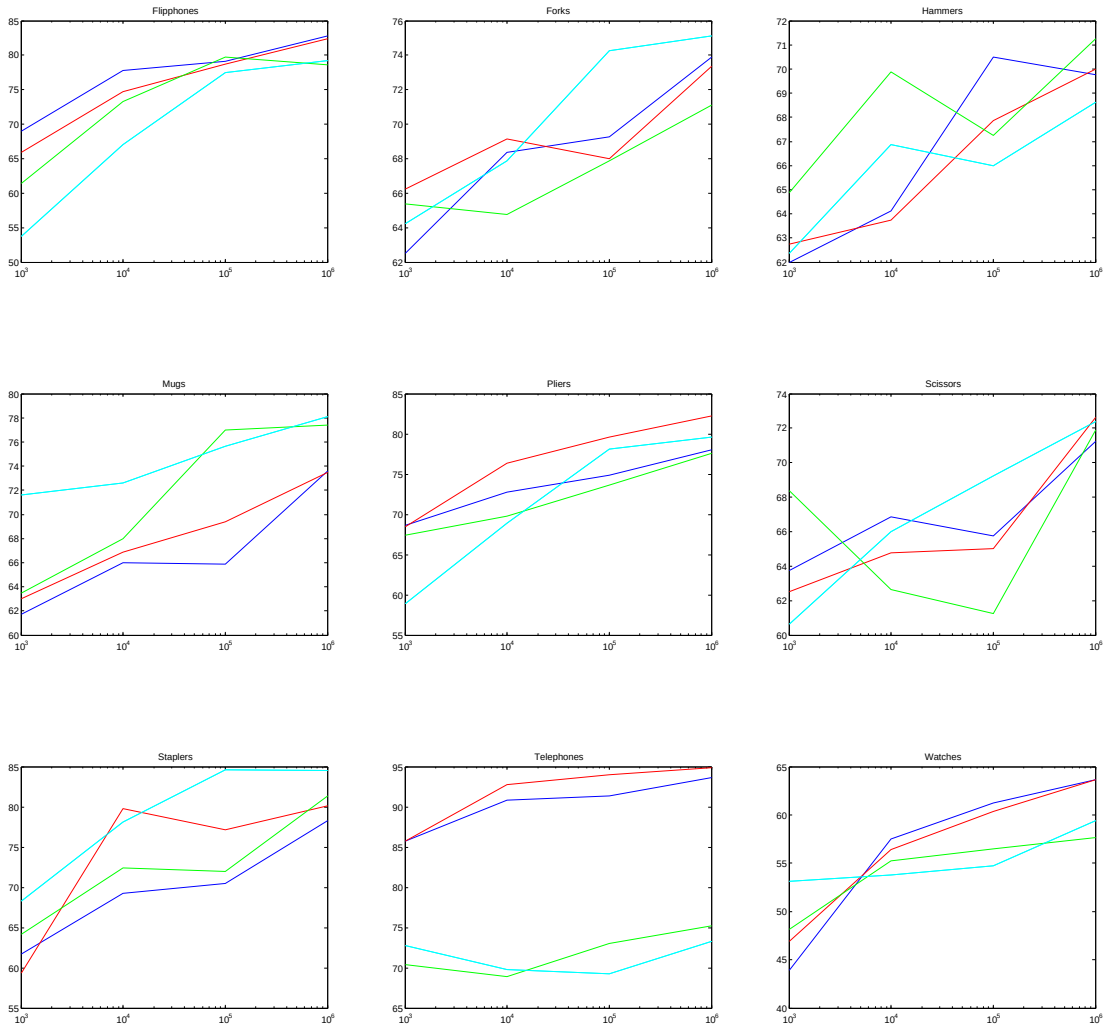

Figure 2: Object detection accuracies for various object classes using cover trees. The x-axis describes the training set size and the y-axis describes accuracy. The colors indicate: Green - Regular 1-NN, Cyan - Regular 10-NN, Blue - Bias Shifted 1-NN, Red - Bias Shifted 10-NN.

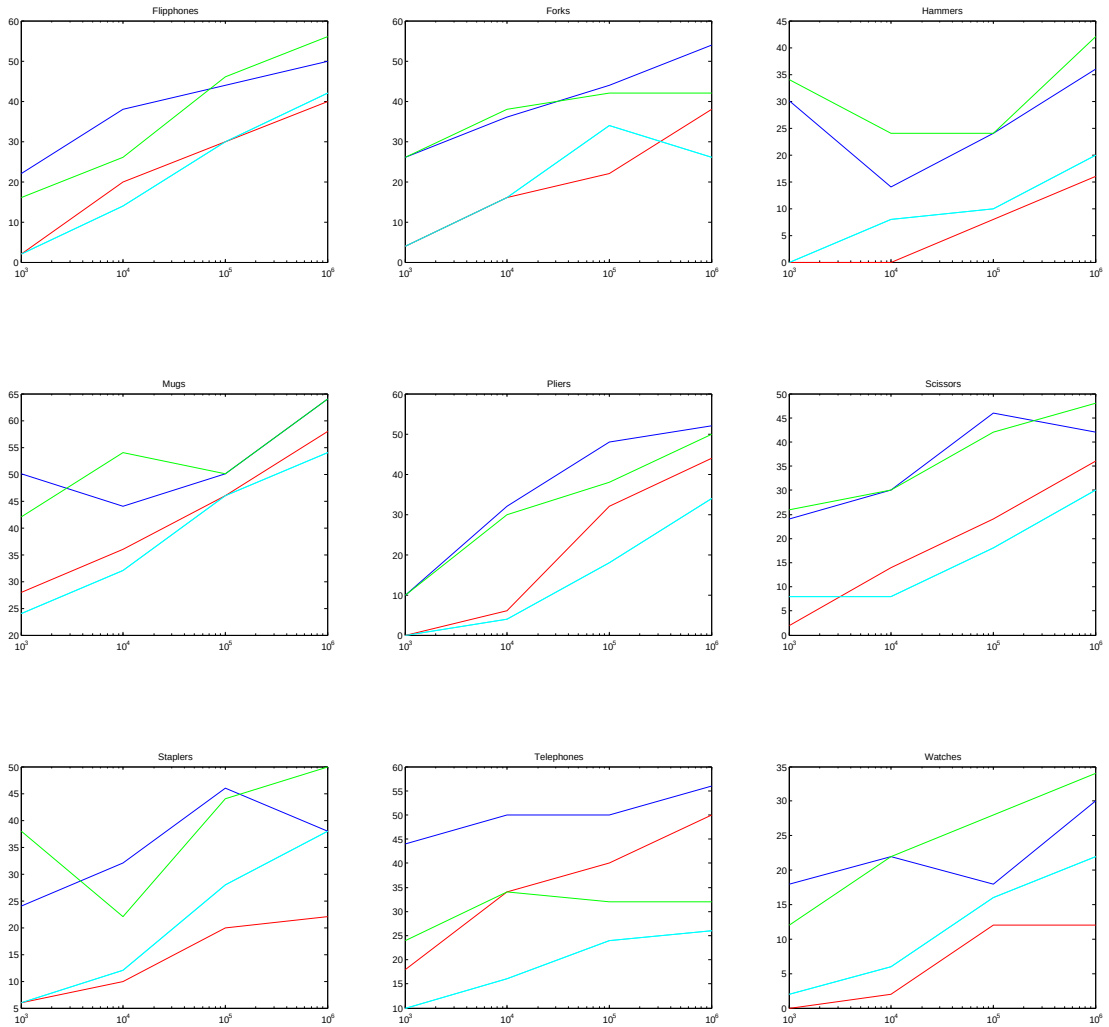

Figure 3: Object classification accuracies for various object classes using cover trees. The x-axis describes the training set size and the y-axis describes accuracy. The colors indicate: Green - Regular 1-NN, Cyan - Regular 10-NN, Blue - Bias Shifted 1-NN, Red - Bias Shifted 10-NN.

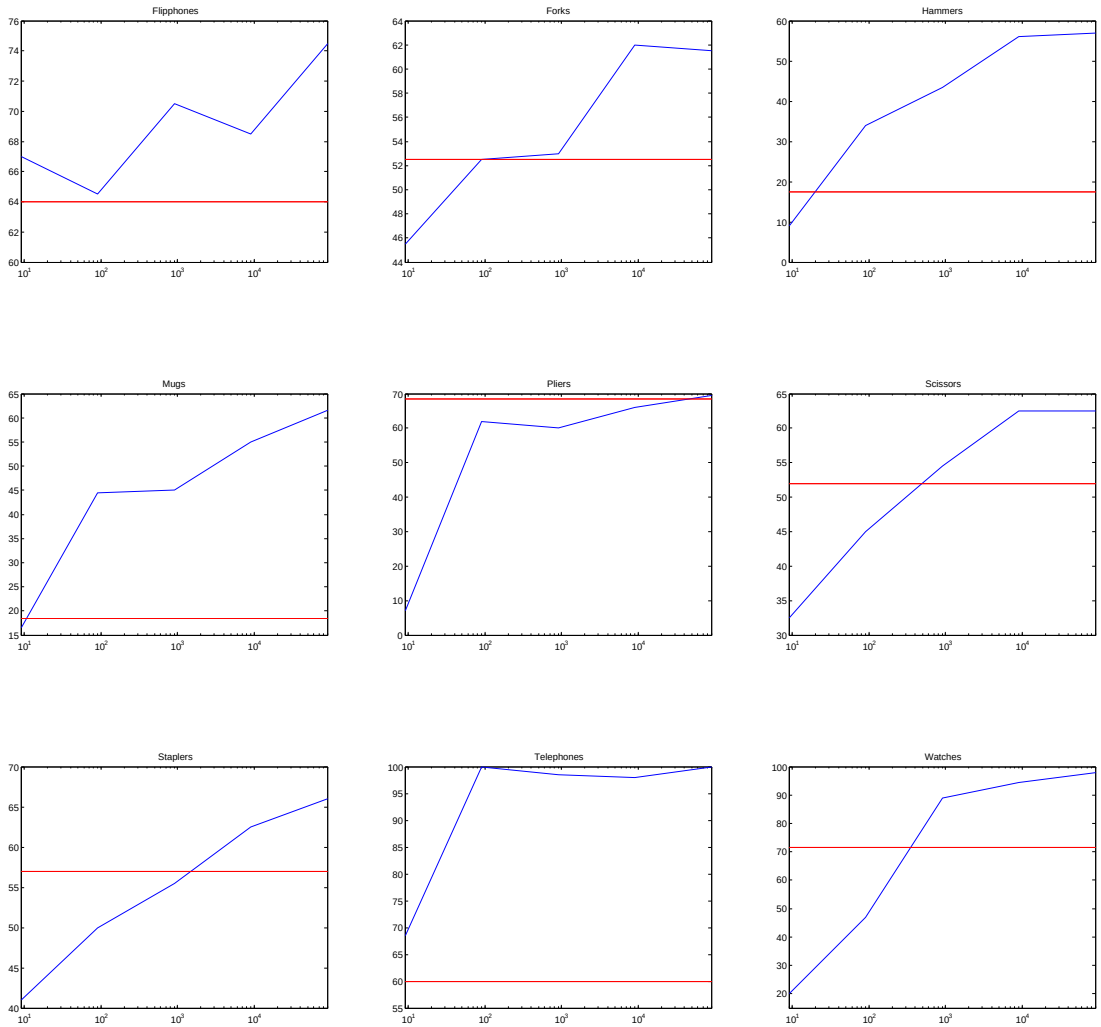

Figure 4: Object classification accuracies for various object classes using Nister trees. The x-axis describes the training set size and the y-axis describes accuracy. The colors indicate: Red - Performance using 1350 green screen object images, Blue - Performance using generated data.