Scene Segmentation of 3D Kinect Images with Recursive Neural Networks

Charles Chen, Jack Chen & Alex Ryan

December 16, 2011

1 Background

In this project, we study scene segmentation of images from the Microsoft Kinect using deep learning techniques. The Kinect gives a depth map of the scene in addition to a standard RGB image, so we are extending methods for scene segmentation and object recognition developed by Socher's group which were previously applied to two-dimensional images. Socher's algorithm parses scenes using recursive neural networks (RNNs) to discover recursive structure in the image.[1] With the availability of a large standardized data set gathered from the Kinect by researchers at NYU,[2] a natural step is to extend Sochers code to apply the RNN-based algorithm to the three-dimensional Kinect images.

Our algorithm starts by oversegmenting the image based on RGB data with the Edge Detection and Image Segmentation (EDISON) system. We compute vision features in these superpixels and map these into a neural network which outputs a "semantic" feature representation for the superpixels. The vision features that we use involve both standard two-dimensional image features as well as features based on the depth information gathered from the Kinect, including measures of the distance and normal vector. The recursive neural network then computes a score indicating whether pairs of adjacent segments should be identified as part of a larger segment, greedily selects the highest score, and computes a feature vector for the new merged segment. This discovers recursive structure for the image segments. The RNN learns a representation for these "super-segments" which allows it to make better merging decisions than with just the original vision features. We use a softmax classifier on the semantic representation to classify the segments.

A paper originally accompanying the Kinect data set published by Nathan Silberman of New York University et al proposed a different method to accomplish a similar task, which involved the computation of location priors of the object classes and different transition potentials to run a CRF algorithm. The method was not incredibly accurate, citing a maximum accuracy of around 56%, citing the "challenging nature of the data." The model also seems hard to generalize because of the necessity for location priors, as well as many tuned parameters and complex feature interdependency.

However, part of the method proposed in the paper is very relevant to our new approach: the data collection method utilized by the Kinect has a few severe artifacts that require nontrivial preprocessing to overcome. The paper discusses the methods used to preprocess the depth information, and this set of preprocessed data is more tractable than the one containing severe artifacts. This preprocessed data is used in our project.

[1]http://nlp.stanford.edu/pubs/SocherLinNg Manning_ICML2011.pdf

[2]http://cs.nyu.edu/ silberman/site/?page_id=27

2 Method

We started with a starter codebase provided by Richard Socher which included a RNN implementation and an initial prediction algorithm based on the average color of each segment in both the RGB and LAB color spaces. We augmented this with several additional features:

2.1 Features

2.1.1 Depth features

Although we were given a small set of test data in our initial code, this did not include depth data captured by the Kinect. We extracted the depth field from the NYU dataset and for each segment, we use as features the mean and standard deviation of depth values in that segment, along with a histogram of the depth values.

2.1.2 Normal vector features

We also computed the unit normal to each segment as a feature. First, the depth field from the Kinect was convolved with a Gaussian kernel in order to smooth out sensor noise. For each point, we calculated the change in depth along the x and y axes and used this to compute the unit normal to the surface represented by the depth data. This operation was vectorized to run quickly in the image pre-processing step. For each segment, we computed the normalized mean of these unit normal vectors as a segment feature.

2.1.3 Haralick texture features

We added several texture features which were first proposed by Robert Haralick based on features of a co-occurrence matrix.[3] The gray levels in the image were discretized into 8 buckets and a co-occurence matrix G was computed with G_{ij} as the number of pixels of value i adjacent to a pixel of value j divided by the number of total pixels. We used 6 statistics calculated from the co-occurence matrix, namely the angular square moment, the contrast, the correlation, the variance, the inverse difference moment and the sum average. This was computed with adjacency defined in the $+x$ direction and the $+y$ direction for a total of 12 features.

[3]http://dceanalysis.bigr.nl/Haralick73-Textural%20features%20for%20image%20classification.pdf

2.2 Principal component analysis

In order to meet memory constraints and improve performance, we reduced the number of features by running principal component analysis on our feature set. After whitening and normalizing each feature to mean 0 and variance 1, we reduced our feature count from 167 components to 70 components. We chose this number of components so that only components with very small eigenvalues were discarded. We also ran tests with varied number of post-PCA features to measure how PCA affected the results.

2.3 Reduced backpropagation on common classes

In our initial results we found that segments belonging to the largest two classes in the data set (background and wall) were predicted too often, and the recall on the less common classes was very poor. The largest two classes had a order of magnitude more segments than each of the remaining classes, which may have drowned out the information on the smaller classes during training. We attempted to avert this issue by reducing the amount of backpropagation on these classes during training.

2.4 Metric & preprocessing

Accuracy is defined as the proportion of predicted pixels that match human-labeled values. In some ways this is a poor metric as confusion of two closely-related classes should be considered at least partially correct. However, we preprocessed our dataset to only use the top 13 classes from the dataset and the NYU group had already unified similar labels with Wordnet.

3 Results & analysis

3.1 Diagnostics & overfitting

We aimed to choose features that generalized to different scenes and were therefore resilient to overfitting to noise in each scene. Still, we found significant overfitting by testing predictions on the training and test data. For PCA to 70 components and full backpropagation, accuracy on the training set was 71.6% while accuracy on the testing set was 52.2%. In addition, the confusion matrices for training data show fairly high recall on most classes, while the confusion matrices on testing data show poor recall for all but the most common classes (see Figure 1).

We compared the results on training on different numbers of images (the test set was the same in all runs). As we increased the training set size, the train set accuracy decreased while the test set accuracy improved, confirming that the poor accuracy is a result of overfitting (see Figure 2).

We expect that the accuracy would have improved further by training on a larger data set, but we did not have sufficient time to run training on more images.

3.2 Quality of features

Figure 3 compares the test accuracy with all the features to the accuracy with a subset of the features, which indicates how much each set of features contributes. The depth and position features appear to contribute a small amount to the accuracy, while texture features did not result in any significant change. Strangely, normal features decreased the accuracy. This may be a result of these features replacing more informative features after running PCA.

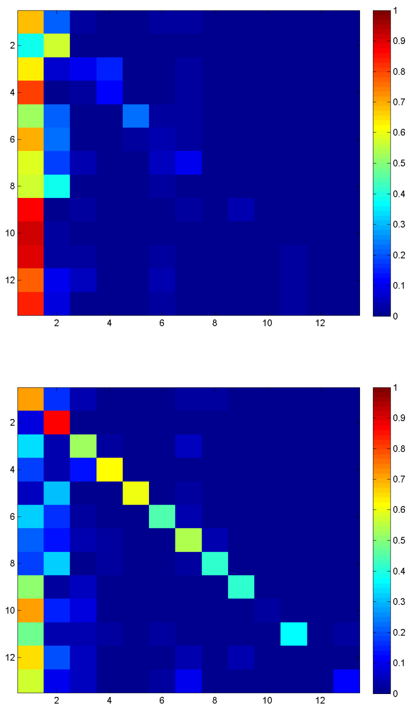

Figure 1: The confusion matrix of test data (top) shows significantly weaker results than the confusion matrix of training data (bottom), suggesting overfitting.

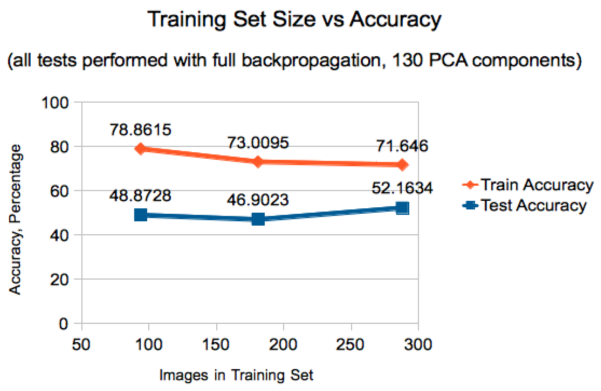

Figure 2: Accuracy of test predictions using all features and 70 PCA components with varying training set sizes.

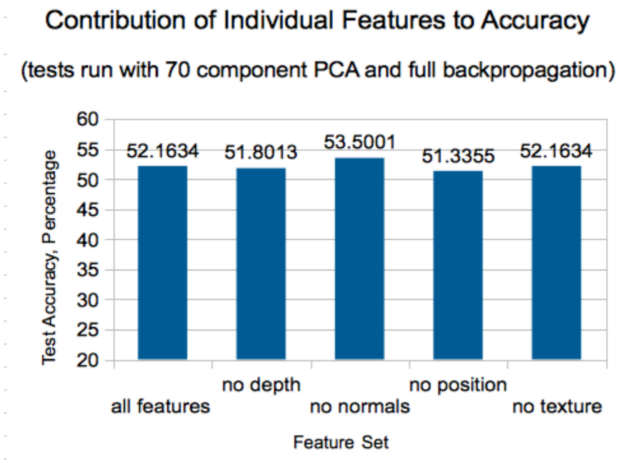

Figure 3: Contribution of individual features to accuracy.

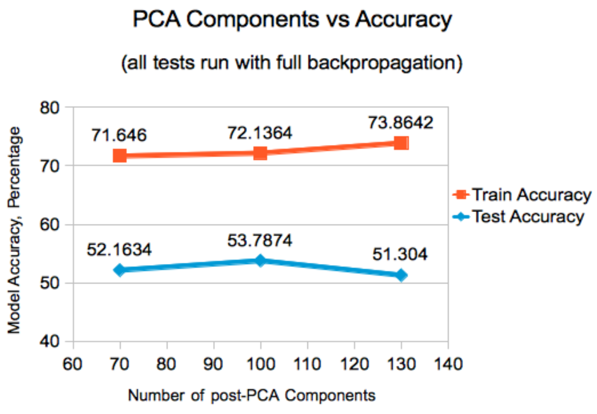

Figure 4: PCA components vs. accuracy for tests run with full backpropagation and all features.

3.3 Analysis of PCA feature count

Figure 4 compares the results of using PCA to reduce the feature space to various numbers of dimensions. With a larger number of post-PCA features, train accuracy was higher, a result of increased overfitting. Test accuracy increased from 70 to 100 components, which suggests that PCA was removing too much useful feature information with just 70 components, but decreased from 100 to 130, which suggests that any useful feature information in these components was outweighed by the increased overfitting.

3.4 Reduced backpropagation on common classes

We varied the backpropagation weight on the two largest classes, while using our full feature set. We found that accuracy on both the test data and the train data did not increase substantially with decreasing backpropagation weight and even decreased when weight was reduced to 0.2 (see Figure 5). Still, examining the confusion matrices for the test data, we found that reducing the backpropagation weight did in fact improve predictions of classes 2 to 13 (see Figure 6). However, because class 1 overwhelmed the other classes in number of segments, the overall accuracy decreased. This is a flaw in the cost metric we used in training our RNN. An alternative approach that may improve results is to modify the cost function to increase the importance of recall on the smaller classes.

Note the slightly improved predictions of classes 2 to 13 with reduced backpropagation weight in figure 6 at the expense of slightly degraded class 1 predictions.

4 Conclusion

We used a recursive neural network algorithm to classify segments of images with a test data accuracy of 53.8%. We ran our implementation with more than 30 different sets of features and other parameters to investigate the efficacy of our features and each part of our model. We would like to acknowledge Richard Socher for providing us with starter code and mentoring us on this challenging project.

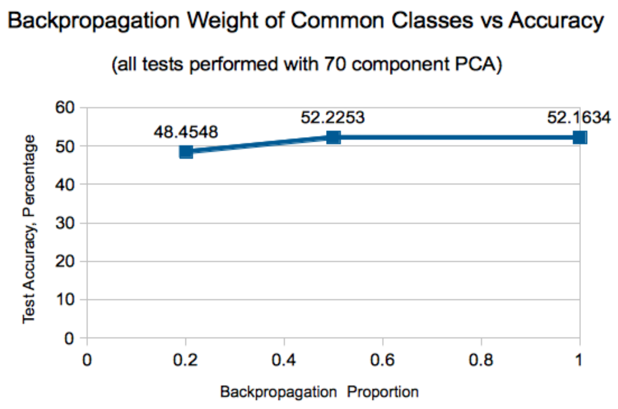

Figure 5: Backpropagation weight of common classes vs. accuracy.

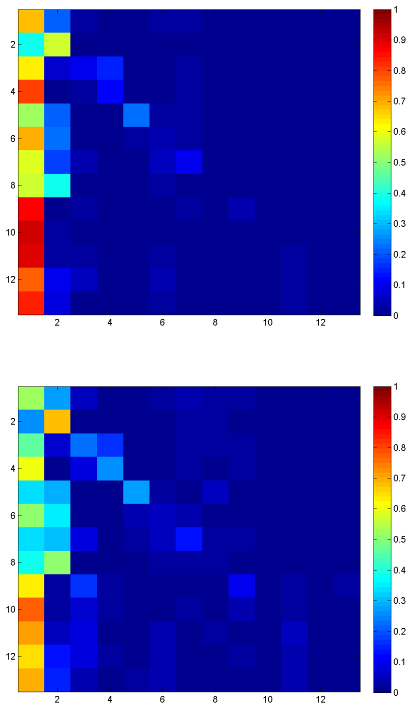

Figure 6: Confusion matrix for backpropagation weight of 1.0 (top) and 0.2 (bottom), each computed for all features and 70 PCA components.