THESIS

# EVALUATING THE ROLE OF CONTEXT IN 3D THEATER STAGE RECONSTRUCTION

Submitted by

Wimroy D'Souza

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2014

Master's Committee:

Advisor: J. Ross Beveridge

Bruce Draper
J. Rockey Luo

## ABSTRACT

### EVALUATING THE ROLE OF CONTEXT IN 3D THEATER STAGE RECONSTRUCTION

Recovering the 3D structure from 2D images is a problem dating back to the 1960s. It is only recently, with the advancement of computing technology, that there has been substantial progress in solving this problem. In this thesis, we focus on one method for recovering scene structure given a single image. This method uses supervised learning techniques and a multiple-segmentation framework for adding contextual information to the inference. We evaluate the effect of this added contextual information by excluding this additional information to measure system performance. We then go on to evaluate the effect of the other system components that remain which include classifiers and image features. For example, in the case of classifiers, we substitute the original with others to see the level of accuracy that these provide. In the case of the features, we conduct experiments that give us the most important features that contribute to classification accuracy. All of this put together lets us evaluate the effect of adding contextual information to the learning process and if it can be improved by improving the other non-contextual components of the system.

## ACKNOWLEDGEMENTS

This thesis would not have been possible without the guidance of my advisors Dr. Bruce Draper and Dr. Ross Beveridge. I would like to thank Dr. Draper, for allowing me to work with the vision lab at CSU, for being such a great mentor and being patient with me while I found my way around. I would like to thank Dr. Beveridge, without whose push my thesis might have taken 14 years to complete. I would also like to thank Dr. J. Rockey Luo for serving on my committee. He was extremely kind to join the committee at the last minute. Also deserving of thanks are Dr. Sanjay Rajopadhye, Dr. Dan Bates and Dr. Anura Jayasumana who were originally on the committee but couldn't make it for the defense. You all have been very helpful when I needed it.

I'd like to thank my vision group lab mates and friends Hrushi, Prady, Som, Jatin and Rahul who have been more like my family at CSU. Thanks guys for helping me keep my sanity while at the same time trying to drive me insane. Thanks also go to my other friends and colleagues at CSU especially the other members of the vision lab - Mo, Maggie and Hao. Seeing them go about doing their work added to my learning experience.

Finally, and most importantly, I'd like to thank my parents William and Mary, my brother Anup and my aunts Juliana and Imelda. Their prayers, blessings and sacrifices let me follow my dream.

TABLE OF CONTENTS

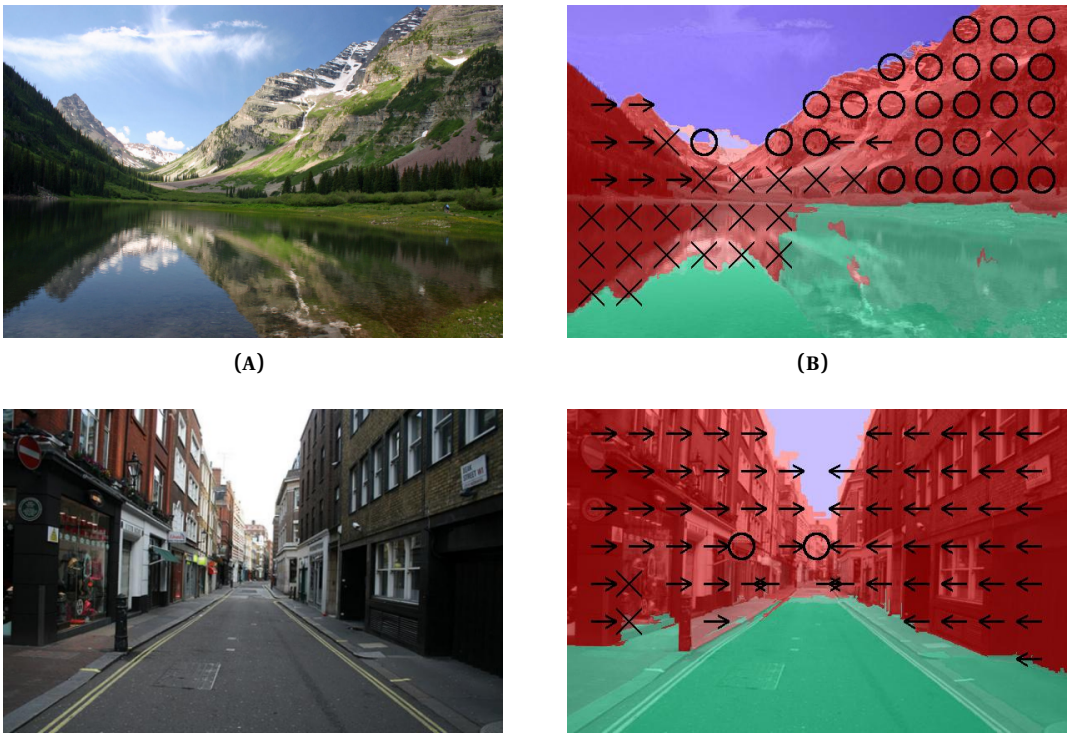
# List of Tables

# List of Figures

CHAPTER 1

# INTRODUCTION

3D theater stage reconstruction of scenes from single images is a problem that finds its origins in the broader topic of scene understanding. Interest in understanding the 3D structure present in still images developed in the early 1960s. But it is only recently that there has been some progress, due in part to the advancement of computing technology and sophisticated algorithms. The term 3D theater stage reconstruction broadly means taking an image and detecting the structure and orientation of the primary surfaces in it so as to be able to build a pop-up model. While there are approaches that use multiple images and stereo imaging, 3D image reconstruction using just a single image is an open problem. Recent work by Hoiem, Efros, and Hebert [10, 11, 12] has tried to fill in this gap. Recovering the 3D structure of scenes has applications such as providing navigation for the visually impaired, an ability to view photographs as though they were taken in 3 dimensions, etc. Moreover, a robust solution to this problem would contribute towards the larger goal of image understanding. Consider, for example, Figure 1.1. The images on the left were obtained using a Google web search, while those on the right were generated after processing them through the system described by Hoiem et al. [12]. We can see that the vertical surfaces (marked in red), and sky (marked in blue) are correctly labeled in both images, and the ground plane (including the surface of water, marked in green is correctly labeled for the most part. The X's and O's indicate solid and porous non-planar objects respectively, whereas the arrows indicate surface orientations. It is quite difficult for artificial vision systems to infer that one view is that of a beautiful mountainside whereas the other is that of a city. Nonetheless, this ability to quite accurately label the



(A) (B)

(C) (D)

**FIGURE 1.1.** Example of labeling mountain-side and city views. Images taken from a Google web search.

various surfaces in an image can be considered a great start in the goal of 3D reconstruction from a single image.

In this thesis we focus on the work by Hoiem et al. [10, 11, 12] on generating a theater stage representation from just a single image. Hoiem et al. [12] use what they call a multiple segmentation approach in order to label regions in an image. This involves segmenting the image multiple times, classifying each region into one of the 3 main classes corresponding to GROUND, VERTICAL and SKY as well as the 5 sub-classes which correspond to LEFT, CENTER, RIGHT (for left, center and right facing planar surfaces) and POROUS and SOLID (for non-planar surfaces). We attempt to analyze the relative importance of context as it is used in the multiple-segmentation approach of Hoiem et al. [12].

(A) (B)

**FIGURE 1.2.** Image of a tree trunk. Images taken from Wikipedia

The word *context* can have multiple meanings. In order to define what *context* means in our case, let us first consider how humans use context to make sense of visual cues. Take, for example, Figure 1.2. Looking at Figure 1.2a it is quite difficult to infer that it is part of a tree trunk. But, looking at the whole picture in Figure 1.2b helps make this inference. Hoiem et al. [12] have used a similar idea in order to infer contextual information from a single image. They partition an image into a number of segments, and then try to infer what each segment represents. They then repeatedly merge neighboring segments that share certain properties, and try inferring what a merged region represents. This is how contextual knowledge is built using just a single image.

The aim of this thesis is to understand the relative importance of features used, classification algorithms and context as used in the multi-segmentation approach of Hoiem et al. [12]. In order to understand the role that context plays in work done by Hoiem et al. [12], we need to answer the following questions.

- Does the choice of a particular classifier matter?
- How good is the performance without additional contextual information?
- Does using neighboring superpixel information improve classification accuracy?
- Which features are the most important and which ones are the least?

- What classification accuracy can we get using just the most important features?

We answer the above questions by conducting a set of experiments as detailed in Section 3.3. These include measuring the contribution of features alone. This will indirectly enable us to quantify the effect that context has on the performance of this system. Since the system described by Hoiem et al. [12] uses a supervised learning approach, it may not work equally well in all scenarios. Hence, we propose to develop a framework where we can exchange system components to cater to a different scenario. We will classify the region features using various classifiers. This will show us if a certain classifier performs better than others or not. We also perform feature selection, in order to gain insight into what features contribute most towards accurately predicting the labels. We use a couple of methods to perform feature selection. This is done to see if there is an overlap between the two methods. We then repeat the classification task to measure the change in performance using the reduced feature set. While we expect the performance to drop a bit, this drop might be acceptable for certain applications where speed is of greater concern.

In Chapter 2, we describe in brief the initial attempts to solve this problem of 3D scene reconstruction as well as some of the alternative current techniques in the literature. We also provide background information needed to understand this thesis. Chapter 3 describes the methodology that we use to ascertain the effect of using only features in 3D image reconstruction. This will help us to know if the added contextual information does indeed improve performance and by how much. In Chapter 4, we present the results and analysis of our experiments. Finally, we conclude in Chapter 5 while also mentioning a few points for future work.

CHAPTER 2

# LITERATURE REVIEW AND BACKGROUND

In this Chapter we first review some of the very first research conducted to infer the 3D structure of a scene from 2D images. We then list some of the more contemporary techniques currently in use. Thereafter, we shift our focus to the approach followed by Derek Hoiem in "Recovering Surface Layout of an Image" [12].

## 2.1. A HISTORICAL PERSPECTIVE

Researchers started looking into the problem of interpreting scenes in natural and artificial images around 1960. A lot of work since thenhas addressed this problem. One of the first works was by Roberts [16] in 1963. In this paper the author treats 2D images as projections of 3D objects with certain assumptions of depth and supporting structures. Guzman-Arenas [8] attempted to separate foreground objects from the background as well as isolate each foreground object into a separate region. This was more in line with the goal of image segmentation that the initial researchers aimed to solve. Brice and Fennema [2] break-down an image into regions of similar gray scale intensity values. This is similar to the concept of generating superpixels as a pre-processing step followed by some of the more modern techniques as mentioned in the next section. This is followed by merging regions with the goal that the resulting regions would have boundaries conforming to natural objects.

One of the most complex image understanding systems developed around early to mid 1960 and whose development continued well into the late 80's was the **V**isual **I**ntegration by **S**emantic **I**nterpretation **o**f **N**atural **S**cenes (VISIONS) by Hanson and Riseman [9]. It's aim was to generate a 3D representation of a scene from a 2D image. This was achieved using – low, intermediate, and high level processes. The low level processes operated at the level

of raw pixel data producing regions and lines. High level processes worked on *aggregates* of the output from low-level processes. The intermediate level processes operated in both a top-down and bottom-up fashion.

## 2.2. Current Work

The development of modern computing technology has facilitated some aspects that were not available to the initial researchers. This includes more computational power and more powerful machine learning techniques. This has led to a supervised learning approach where a system is trained using a large corpus of images and then newer images are analyzed using prior knowledge.

One of the more modern works that broadly follows this approach is by Gould, Fulton, and Koller [7]. In this work, Gould et al. [7] divide an image into regions and then label each region using appearance and geometry. In this work, the authors classify image regions into horizontal, vertical and sky as well as sub-categorize them with a semantic label. While the authors use large regions in an image to label, these regions are typically larger than superpixels and are allowed to grow to conform to object boundaries. This method iteratively optimizes an energy function whose parameters depend on region appearance as well as inter-region potentials. Raw features calculated over a pixel neighborhood include a 17-dimensional color and texture vector. These are fed to a boosted classifier whose output is then appended to the raw pixel feature vector, which captures local appearance information. To capture global appearance information they calculate individual and inter region potentials that include shape, area, moments and boundaries.

Another recent example of an attempt to solve the problem of 3D reconstruction from a single image is by Saxena, Chung, and Ng [17] and Saxena, Sun, and Ng [18]. Saxena et al. [17]

use a multiscale Markov random field (MRF) to model depth in a single image. They focus on both, monocular as well as stereo cues in a supervised learning approach to estimate the depth. Ground truth is calculated using a 3D-laser scanner that returns depthmaps. Absolute depth, according to the authors, models "local feature processing" in the human visual system. They estimate the absolute depth using 9 Laws' masks and 6 oriented edge detectors and use 2 color channels to model texture and haze. Initially, they divide an image into rectangular regions instead of generating an over-segmentation and compute 17 values for each such region using the above mentioned filters. Also, since local features would be poor estimators of depth, they use these features at varying scales in order to capture more information. This is especially the case for features related to texture. They also use information from neighboring regions resulting in a 646 dimensional feature vector. Saxena et al. [17] also estimate the relative depths which model how humans associate if a surface is part of another. For this they generate a 170 length feature vector obtained by creating a 10-bin histogram of the 17 filter outputs mentioned above. They then use two MRFs to model relative depths. One is a Gaussian MRF and the other uses a Laplacian instead of a Gaussian. The Laplacian MRF is more robust to outliers than a Gaussian MRF since it has longer tails and also can model edges better.

But the focus of our thesis, is the work by Hoiem et al. [12, 11, 10]. In Hoiem et al. [10] they lay the foundations of recreating the 3D structure of a scene given its 2D representation as a single image. In this work, Hoiem et al. [10] used features like location, shape, color (in RGB and HSV color spaces), texture and 3D geometry in order to build a pop-up model from an image using a multiple segmentation framework. The complete feature set is listed in Table A.1. The authors aimed to separate regions of an image into the following surfaces – GROUND, VERTICAL and SKY. The vertical surfaces are then popped up to give a virtual reality

like feel. The two main components of Hoiem et al. [12] are the selection of features and the addition of contextual information using a multi-segmentation approach. Each of these are described in more detail in the following section.

2.2.1. HOIEM ET AL. IN MORE DETAIL. While Hoiem et al. [12] use location, shape, color (in RGB and HSV color spaces), texture and perspective features, this feature set differs from the one used in the previous work Hoiem et al. [10]. The complete set of new features are listed in Table A.2. Some features are seen in both papers implying that Hoiem et al. [12] found them more relevant. Exactly how important they are in obtaining a given classification accuracy, will be seen in the experiments section. The main differences in the feature sets between Hoiem et al. [10] and Hoiem et al. [12] are that the earlier work uses Derivatives of Oriented Gaussians (DOOG) filters as well as textons to extract texture information from the image, where as the more recent one uses a subset of the Leung-Malik filter bank for the same purpose. The other features are more or less the same with location, shape, line and intersecting lines based features appearing in both feature sets. With respect to obtaining a better 3D perspective, the recent paper has also added vanishing points.

Hoiem et al. [12] use a multi-segmentation approach in order to estimate the labels associated with each segmentation of the image. Multi-segmentation in this case means generating a particular segmentation of an image and then merging some of the regions resulting in another segmentation of the same image. This allows regions of differing sizes to be analyzed. This is how contextual information is added to the analysis. A region that is consistently labeled across multiple segmentations is more likely to be labeled correctly. It is also seen from the results in Table 2.2 that context used in the multiple segmentation framework does indeed play an important part in image understanding. But no data is available for a per-class

**TABLE 2.1.** Confusion matrices for multiple segmentation method [12]

**(A)** Main classes

| | Support | Vertical | Sky |
|---|---|---|---|
| Support | **0.84** | 0.15 | 0.00 |
| Vertical | 0.09 | **0.90** | 0.02 |
| Sky | 0.00 | 0.10 | **0.90** |

**(B)** Sub classes

| | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|
| Left | **0.37** | 0.32 | 0.08 | 0.09 | 0.13 |
| Center | 0.05 | **0.56** | 0.12 | 0.16 | 0.12 |
| Right | 0.02 | 0.28 | **0.47** | 0.13 | 0.10 |
| Porous | 0.01 | 0.07 | 0.03 | **0.84** | 0.06 |
| Solid | 0.04 | 0.20 | 0.04 | 0.17 | **0.55** |

improvement with the addition of context. As seen in Table 2.1 we are only provided with the results of the complete approach. What we have is an average accuracy across all labels.

To the best of our knowledge, there hasn't been any study on the effect of context as used in the multi-segmentation approach in the 3D scene reconstruction process. One paper that comes close to explaining context is by Divvala, Hoiem, Hays, Efros, and Hebert [4] which is primarily an attempt to define what *context* means as it is used in the literature. According to Divvala et al. [4], different researchers use the word *context* with differing connotations and this lack of a definitive meaning assigned to the word prevents a greater understanding of how well an approach suits a given problem. For example, Divvala et al. [4] mention that we could have geometric context, semantic context, geographic context, temporal context and so on. Use of each distinct type of contextual information would be useful in certain tasks more than others. Also, since this is an empirical study [4], we are trying to see if the experiments in Hoiem et al. [12] are reproducible. We try to evaluate how one component works independent of the others. Below we broadly enumerate the various steps used by Hoiem et al. [12].

**TABLE 2.2.** Average accuracy of various methods [12]

| Method | Main | Sub |
|---|---|---|
| Pixels | 82.1 | 44.3 |
| Superpixels | 86.2 | 53.5 |
| Single segmentation | 86.2 | 56.6 |
| **Multiple segmentation** | **88.1** | **61.5** |
| Ground truth segmentation | 95.1 | 71.5 |

(1) Generate superpixels using Pedro Felzenszwalb's graph-based segmentation algorithm[6]. This algorithm takes an image as input and generates a segmented image with a random color assigned to each segment. An example is shown in Figure 2.3.

(2) The next step is to compute some elementary properties or features as listed in Table 3.2 for each superpixel in the image. These features are then used as input to separate decision tree classifiers which return the likelihood/probability of each superpixel belonging to one of the main classes as well as the sub-classes.

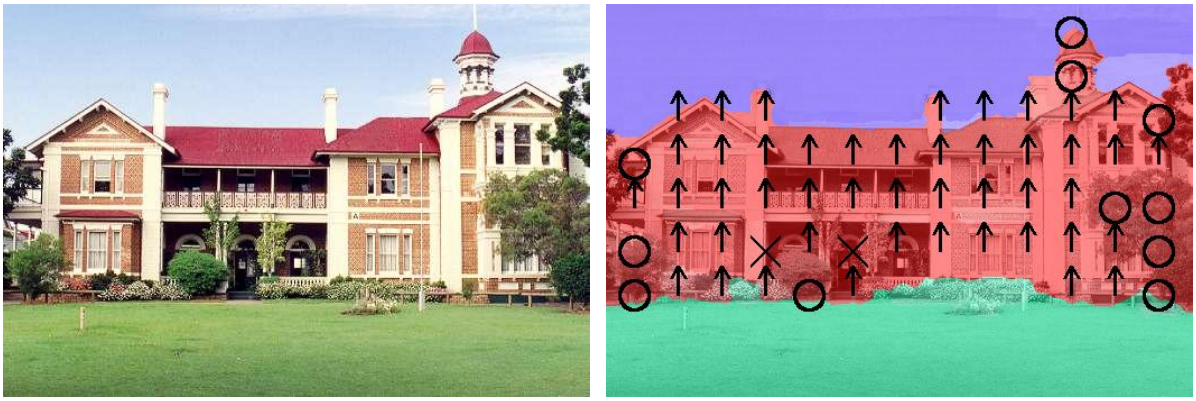
(3) Calculate image-level features such as vanishing points and horizon estimates.

(4) Combine superpixels into a varying number of segments per image and then for each one calculate the features listed in Table A.2.

(5) Estimate the labels for each super pixel and segment and assign superpixel labels based on a weighted measure from all segments to come up with a final score. This approach assumes different segmentations give errors in different parts of the image and therefore these errors will average out over a number of segmentations.

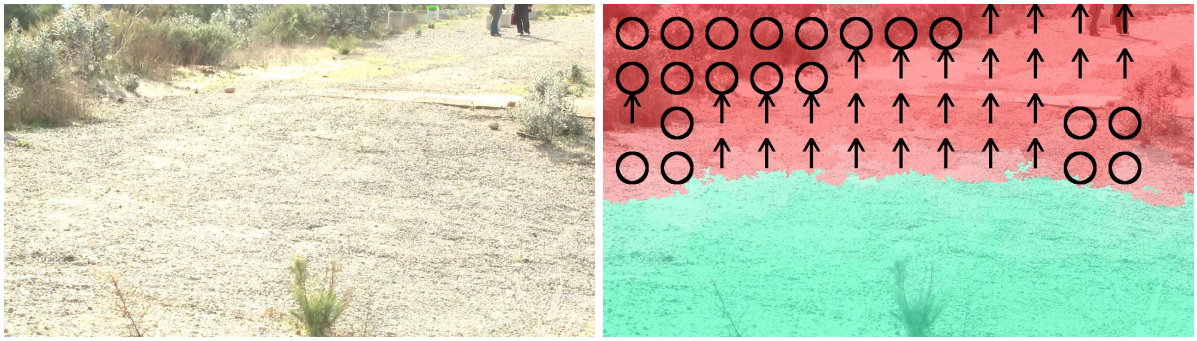
(6) Once done, the processed image looks like Figure 2.1b.

To see an example of when the system fails to classify accurately, see Figures 2.2a and 2.2b. Note, however, that the porous parts in the image represented by small bushes and shrubs are quite correctly classified. But it wrongly detects a horizon, in the center of the image and



(A) Original image (B) Final processed image

**FIGURE 2.1.** Image labeling and classification



(A) Image which does not have a well defined horizon (B) The resulting poor classification

**FIGURE 2.2.** Example of bad classification

classifies the upper half, which is actually GROUND, as VERTICAL. This would result in a lower accuracy in labeling the main classes, but the sub-class POROUS would have a high score.

## 2.3. ADDITIONAL BACKGROUND

This section provides some additional background information on superpixels which are becoming an important pre-processing stage in computer vision systems.

2.3.1. SUPERPIXELS. Images and video frames are made up of picture elements or pixels as they are commonly called. While the whole image frame might depict an interesting scene, each pixel by itself is rather meaningless. Moreover, trying to gauge the significance

of a pixel by considering its neighbors also doesn't provide as much information as we might like. This is where researchers turn to something called superpixels. Superpixels are nothing but a collection of pixels that share a certain similarity. The process of aggregating pixels into superpixels is done using some function that helps collect similar and contiguous pixels into a larger region. This larger region encodes much more information than that could be provided by just using pixels. While the act of generating superpixels might seem similar to a closely related technique called image segmentation, their aims are different. Superpixel generation also called image over-segmentation aims to assign labels to pixels such that pixels sharing similar properties of color, texture etc., are grouped together. In contrast, image segmentation aims to partition an image into semantically meaningful regions. A segmentation procedure attempts to create regions that adhere to object boundaries and, in the ideal case, there should be one whole object per region. The use of the word *segmentation* differs from its use in the multiple segmentation based methods [11, 12] where each segmentaion implies a different collection of superpixels obtained by merging neighboring superpixels that share some similarity.

Superpixels enable us to compute some elementary image statistics that encode localized information. They also provide a convenient method to process similar regions in an image instead of processing each pixel individually. This results in a substantial reduction in the time required to process an image. Superpixels can be generated using one of the many over-segmentation algorithms [6, 1, 19]. The various over-segmentation algorithms, while trying to divide images into similar regions, take slightly differing approaches. For example, Simple Linear Iterative Clustering (SLIC) [1] is based on k-means clustering and takes the number of superpixels needed and initializes that many cluster centers uniformly across the image. The

(A) Original Image (B) Graph-based image segmentation (C) Segmentation using SLIC

**FIGURE 2.3.** Example of Felzenszwalb's segmentation

number of superpixels finally generated can be less than the number requested. SLIC produces mostly uniform superpixels depending on the parameter passed. Felzenszwalb's graph-based approach [6] aims to form a superpixel over as much of a uniform area as possible. But this too could be tuned, although a bit more tediously. Example segmentations are shown in Figure 2.3.

Hoiem uses the Felzenszwalb's graph-based over-segmentation algorithm [6] as the first stage of the pipeline. We could try replacing this algorithm with another one such as SLIC, but for the time-being we haven't chosen to do so. Figures 2.3b and 2.3c show the differences between two over-segmentation algorithms. We can see that SLIC superpixels are more regularly shaped.

CHAPTER 3

# METHODOLOGY

In this Chapter, we outline in brief our contributions to the understanding of Hoiem's work. We begin by describing the background of the various components used in further detail, followed by a description of the experiments conducted in order to better understand the work on scene understanding by Hoiem et al. [12]. Specifically, we aim to evaluate the role of context in solving the image understanding problem. Recall that context here is a very specific term discussed in Chapter 2. It involves segmenting an image into superpixels, classifying each superpixel, combining similar superpixels into larger regions, classifying each larger region and assigning the most likely label.

In order to reason about a scene depicted in a photograph, we must first be able to reason about the physical structures like buildings, trees, people, cars, etc., present in the scene. Then, we can go on to find the inter-relations between these structures. But to find the relative positional relationship between the structures we most importantly need a ground plane, also called a support structure. This is true even for humans, who wouldn't be able to gauge relative distances of far away objects without a ground plane [14]. It is usually this plane which even humans use as a guide to estimate depth beyond the human eye's stereoscopic ability, which works only at short distances. At the moment, human and machine vision operates in an environment with gravity. Most often objects rest upon the ground or some other supporting structure. Thus, we can define the *ground plane* as any horizontal surface or horizontal plane providing a support to the vertical structures. It is the horizontal plane on which we see people standing or walking. The farthest limit of this plane would be the set of points defined by the horizon in the scene. In addition to the ground plane and vertical structures

Hoiem et al. [11, 12] also attempt to label sky. In addition, the vertical structures are further sub-classed into planar surfaces facing left, center and right and non-planar surfaces that are either solid or porous.

In order to evaluate the effect of context on solving the problem of understanding scene structure, we conduct experiments without the added contextual information. This will show how much the added contextual information helps using the multiple segmentation approach. In the following sections we detail the background information required to perform a quantitative analysis, in particular the dataset and features, as well as a list of experiments to be performed and the associated evaluation criteria.

## 3.1. Dataset and Ground-truth

The dataset used is the *CMU Geometric Context* dataset used by Hoiem in [11, 12]. It consists of 300 publicly available images of outdoor scenes. These images are a varied mix of alleys, buildings, shorelines, etc. Some of the images are shown in Figure 3.1. Hoiem labeled this data using the sets of labels descriibed in Chapter 2. Recall, those labels are GROUND, VERTICAL and SKY for the main class and LEFT, CENTER, RIGHT, POROUS and SOLID for the sub class.

Ground-truth data for the segmented images has been provided by Hoiem et al. [12]. There are some unavoidable inconsistencies due to the labeling task being quite subjective. Hence, estimation accuracy needs to account for ground truth mis-labeling as well. These errors are more common in sub-class labeling than the main-class labeling. The ground-truth data includes the segmented image labels, number of superpixels, labels for the main and sub-classes, superpixel adjacency matrix, image name and image size. Some properties of the dataset are listed in Table 3.1. In carrying out this work we chose to base our experiments on

**TABLE 3.1.** Dataset Properties

| | |
|---|---|
| Number of images | 300 |
| Total number of superpixels | 151,576 |
| Feature vector length | 52 |
| Number of images with unlabeled segments | 13 |
| Number of unlabeled segments | 192 |

**TABLE 3.2.** Superpixel features[12]

| | |
|---|---|
| L1 | Location: normalized x and y, mean |
| L6 | Shape: normalized area in image |
| C1 | RGB values: mean |
| C1 | HSV values: C1 in HSV space |
| C3 | Hue: histogram (5 bins) |
| C4 | Saturation: histogram (3 bins) |
| T1 | LM filters: mean absolute response (15 filters) |
| T2 | LM filters: histogram of maximum responses (15 bins) |

the dataset and ground-truth provided by Hoiem et al. [12] because using their data would help in a direct comparison with their results. Figure 3.1 gives us an idea of the type of images in the dataset.

## 3.2. FEATURES

Hoiem et al. [12] use 94 features in all, of which 52 are used to label only the superpixels, where as the complete set is used in the multiple-segmentation stages. We can't use the complete set with superpixels because the additional features work on a whole image. For example, vanishing points can not be calculated at the level of superpixels. The first stage in the multiple-segmentation approach is to generate superpixels. Table 3.2 lists the features used at the first stage. The complete list of features used are shown in Table A.2 and these are used across larger regions once the use of spatial context using multiple segmentations comes into the picture.

Color, location and texture information is used at this stage. Texture features are calculated using a subset of the Leung-Malik filter bank [13]. The original Leung-Malik (LM) filter bank

(A) Alley

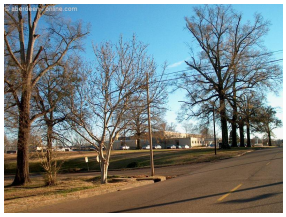
(B) City

(C) College

(D) Fields

(E) Flat

(F) Lawn

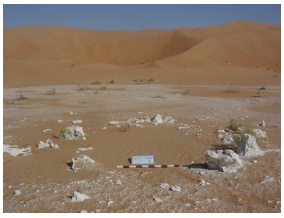


(G) Neighborhood

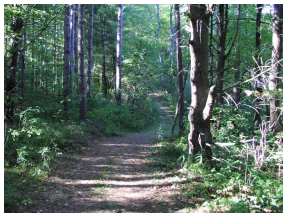
(H) Outdoor

(I) Scenery

(J) Shore



(K) Structure



(L) Trail

(M) Urban

**FIGURE 3.1.** Dataset Example [12]

contains 48 filters – 36 elongated filters at 6 orientations, 3 scales, and 2 phases, 8 center-surround difference of Gaussian filters, and 4 low-pass Gaussian filters. Hoiem et al. [12] use

a slightly modified subset (15) of the LM filter bank. Specifically, they use – 6 edge and bar filters, 2 Laplacian of Gaussians and 1 Gaussian. Figure 3.2 provides a visual representation of these. The scale factor is chosen as $\sqrt{2}$ and the filters operate in a 19x19 pixel neighborhood. The features calculated from this bank are the mean of the absolute responses of each filter and a histogram of maximum responses, each of which contribute 15 features to the overall feature vector.

## 3.3. Experiment Design

The following subsections specify the experiments designed to answer the questions asked in Chapter 1 viz., the effect of removing context, adding local neighborhood information comparing classifiers, and evaluating which features are really important.

3.3.1. Evaluating Classifiers. This experiment quantifies the performance of various classifiers on the CMU Geometric Context dataset. Given a feature set, this will let us know how one classifier performs with respect to the others. If all classifiers tested give similar labeling accuracy, we can conclude that classifier selection doesn't matter as the features themselves have a lot of discriminative power. Moreover, different implementations of the same type of classifier could have different performance characteristics.

Hoiem [11, 12] used boosted decision trees as a classifier. Boosted classifiers work on the principle that, a group of weak classifiers used together can combine to generate a strong classifier. Each classifier in the sequence aims to correct the mistakes made by the previous classifier. Hoiem et al. [12]'s work doesn't mention the effect of using some other classifier, but suggests that a better classifier could produce better results. In our evaluation we use 3 classifiers - two implementations of an SVM classifier and one decision tree classifier. We use an out-of-the-box Support Vector Machine (SVM) classifier in order to generate baseline results

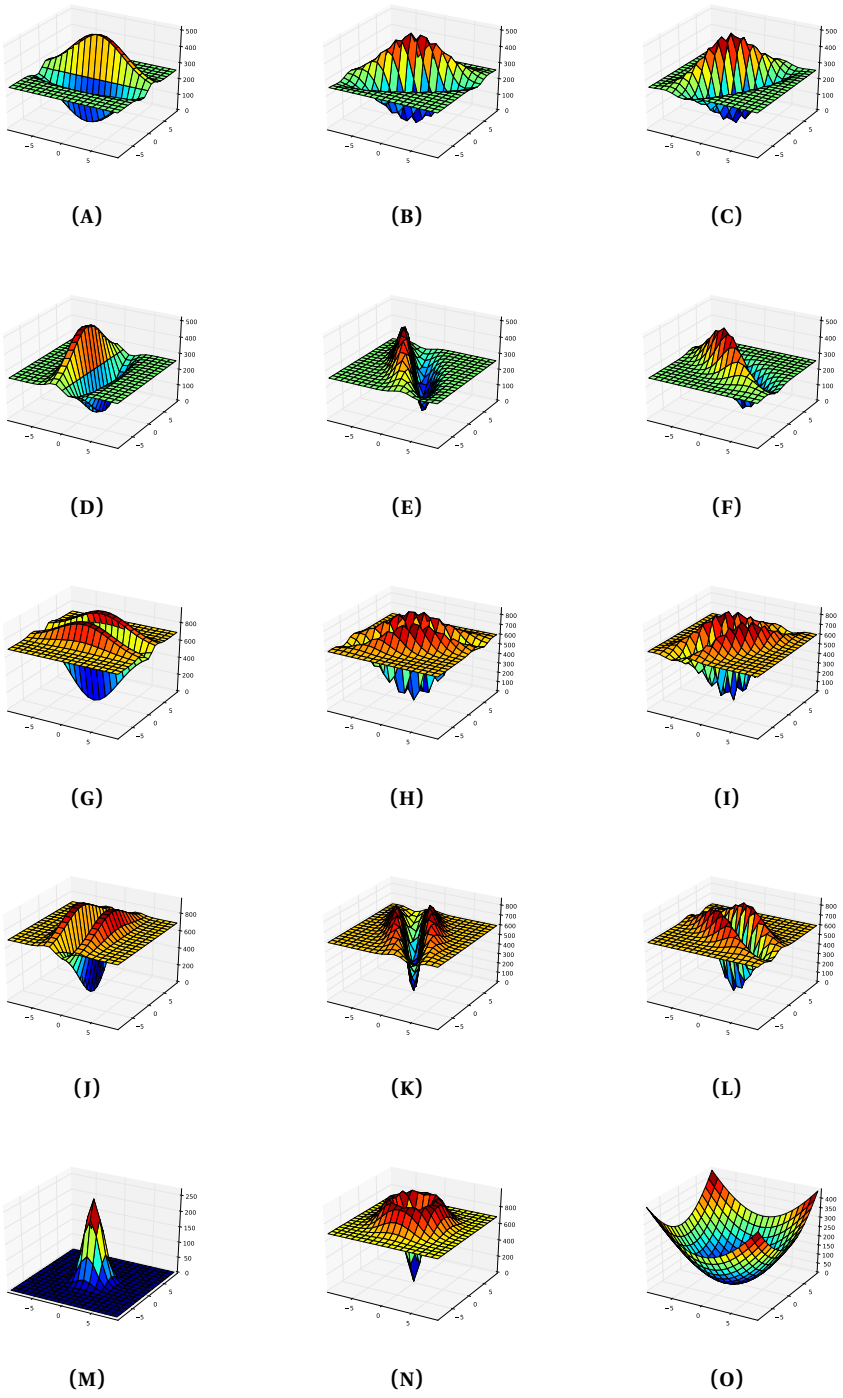



**FIGURE 3.2.** Leung-Malik Filter Bank. Filters (a) through (l) are derivatives of Gaussians at various scales and orientations, (m) is a Gaussian and (n) and (o) are Laplacian of Gaussians.

against which the other 2 would be evaluated. In this case, we use the freely available software package "Liblinear" [5]. This package implements an SVM using a linear decision function. We

choose this classifier instead of the other popular implementation "LibSVM" [3] because the implementation of the linear kernel is much faster in liblinear than in libsvm, on the order of a few days. For example, training along with 5-fold cross validation using the features from 200 images takes about a week using libsvm, where as liblinear takes a few minutes at most. Libsvm using the linear kernel also doesn't scale well for large datasets. Moreover, there isn't any substantial gain in classification accuracy using libsvm with the linear kernel.

We also evaluate the performance of two other classifiers – sklearn-liblinear and sklearn-decision-tree, which are implementations of liblinear and decision trees in the sklearn software package [15]. Using sklearn-liblinear helps us to capture differences between two implementation of an SVM classifier. We show confusion matrices for each of the classifiers tested and these would help us know if there is any substantial difference in using one classifier over another.

The procedure and evaluation criteria is as follows. We first split the 300 images into 5 sets of 60 each and then perform 5-fold cross validation. This is done multiple times and the average labeling accuracy is shown in the form of confusion matrices and box plots. Both these together show us how one classifier performs relative to another. We further set the evaluation criteria to be a change of 5% or more. What this means is, if classifier A gives an average accuracy of 70% for a given label and classifier B gives 75% then this is a significant difference. Anything less that 5% does not matter. We set this criteria on a per-label basis because of the assumption that certain features may be conducive to better classifying certain labels.

3.3.2. USING NEIGHBORHOOD INFORMATION. This experiment quantifies the change associated with changing a label based on its neighboring pixels. This is a very simplistic approach and may only help correct for inconsistencies such as, where a patch of ground is

surrounded by sky. In this case, we can change the label SKY to GROUND with reasonable certainty. But in the case of the sub-classes this isn't necessarily the case. For example, a left facing surface could be surrounded by center or right facing surfaces. Porous objects could be surrounded by solid objects and so on. On the other hand superpixels that are correctly labeled in the first place could be changed incorrectly resulting in a drop in classification accuracy. We then modify this simplistic approach, by changing the label based on certain other considerations enumerated below.

(1) Change the label if it is surrounded by a majority of differing labels ($C_{majority}$).

(2) Change the label only if the absolute difference between the given label and the majority of its neighbors is 1 ($C_{neighbors}$).

(3) Change the label only if the superpixel size is less than a certain value ($C_{size}$).

(4) Change the label only if the probability of its classification is within a certain threshold ($C_{threshold}$).

We show plots for each of the 4 cases listed above as compared to the baseline. We do this for the average case as well as for the per-label case. If there is a change of 5% or more then we consider that change to be substantial.

3.3.3. FEATURE SELECTION. Experiments in this section answer the question of which subset of features give a high labeling accuracy. While it is almost certain that using a subset of the original 52 features would result in a lower accuracy, how much lower is what this experiment lets us know. We find out the feature subset using 2 different methods of feature selection. One of these methods is recursive feature elimination (RFE). The other is using a forest of trees. We use 2 methods for feature selection to ascertain if there are features which rank highly in both the methods. Features that rank highly in the 2 methods will imply that they are indeed important. These 2 methods are described below.

3.3.3.1. *Using Recursive Feature Elimination.* Recursive Feature Elimination (RFE) is a method for selecting a subset of the most important features that contribute to classification accuracy. As the name suggests, we perform this procedure recursively and at each iteration remove the features that made the least contribution to accuracy. In order to decide which features need to be eliminated, we need a weighting function. Such a weighting function will assign weights to each feature based on its contribution and allow us to remove the features with the lowest weight. In this experiment, we use an SVM classifier, whose coefficients are used as the weighting vector. The algorithm takes the output of this classifier, and at every iteration removes features with the lowest weights. For this experiment, we use a number of images ranging between 10 and 80 in increments of 10. This will tell us if having more images in the training set has any bearing on the selection of features.

3.3.3.2. *Using Forest of Trees.* We can use decision trees or a collection of trees known as a forest for the purpose of feature selection. Decision trees operate on the principle of making simple decisions with the aim of maximizing the split in data classes at every node in the tree. They also attempt to keep a node as pure as possible. There is usually, however, some impurity at most of the higher nodes in a tree, resulting in a need to grow the tree further to separate the impure clusters. At each such decision point, there is an associated score assigned to the feature used to make the decision. The higher this score, the better the given feature helps with classification. This score is what is used to identify the most important features in a dataset with high dimensionality. The sum of scores across all features is 1. In this method we use images in sets of 100, 200 and 300 to do the evaluation. Each set of images gives us a number of highly ranked features. The ones that are common near the root of the tree should be considered to have more discriminitive power.

3.3.4. CLASSIFICATION USING THE REDUCED SET OF FEATURES. Our final set of experiments involves classifying the segmented regions using just the subset of features obtained from the experiments conducted as per Section 3.3.3. Our aim is to know the increase in labeling errors the classifiers make using the reduced feature set in place of the full feature set. There are many reasons for using a reduced subset of features. It reduces the dimensionality of the data as well as the computation time required by the classifier. There may be some applications where this trade-off relative to a slight loss of accuracy would be acceptable to the reduced computational time. Again, we plot the classifier accuracy using the reduced feature set against the full feature set. We use only 1 classifier for this experiment - an SVM (liblinear). This procedure is evaluated using 5-fold cross validation with comparisons plotted across 20 runs. We use the reduced feature set obtained by both the feature selection methods described in Section 3.3.3.

CHAPTER 4

# RESULTS

## 4.1. EXPERIMENTAL RESULTS

The sections below present the results of our experiments on evaluating the classification accuracy for the main classes, viz., GROUND, VERTICAL and SKY as well as the sub-classes, viz., LEFT, CENTER, RIGHT, SOLID and POROUS. These results help us understand the role that context plays in understanding the scene structure for each case.

4.1.1. BASELINE. In the first experiment we set out to establish our baseline by using only the superpixel features to classify the main classes using a Support Vector Machine (SVM) classifier. The dataset includes the 300 images from the CMU GeometricContext dataset and its associated ground-truth. We randomly split the dataset into 5 parts, each containing 60 images and run 5-fold cross-validation, using 4 parts for training and 1 part for testing the accuracy. Since there are approximately 150,000 samples, there are on average 30,000 samples in each fold. Our criteria for evaluating classifiers would be to see if there is at least a difference of 5% on a per class basis. The 5-fold cross-validation is done 20 times and the mean-accuracy across these runs is calculated. For simplicity, we split our analysis for the main and sub classes below. Also, we only show the row-normalized confusion matrices. Detailed tables with raw numbers are provided in appendix B.

4.1.1.1. *Main Classes.* Table 4.1 shows the row-normalized confusion matrix. Row normalized tables are calculated by summing across rows of the confusion matrix and then dividing each row by its sum and then multiplying by 100 to get percentage values. Here, UNLABELED is a class that has no bearing on our analysis. Some superpixels were found to be unlabeled in the ground-truth data. There are in all 192 unlabeled superpixels out of a total of over 150,000

samples. We also show the corresponding mean and variances across 20 runs of 5-fold cross-validations as a box-whisker plot in Figure 4.1. We can clearly see that the performance for the VERTICAL class is much higher and with a smaller variance than that of the other two main classes.

**TABLE 4.1.** Confusion matrices for main classes using liblinear

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 12.84 | 26.74 | 36.95 | 23.46 |
| Ground | 0.02 | 69.63 | 29.70 | 0.65 |
| Vertical | 0.00 | 9.05 | 89.53 | 1.42 |
| Sky | 0.05 | 2.57 | 33.06 | 64.33 |

**TABLE 4.2.** Confusion matrices for main classes using sklearn-liblinear

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 7.89 | 22.14 | 34.79 | 35.18 |
| Ground | 0.05 | 70.59 | 28.39 | 0.98 |
| Vertical | 0.01 | 8.95 | 89.54 | 1.49 |
| Sky | 0.14 | 2.51 | 27.66 | 69.69 |

**TABLE 4.3.** Confusion matrices for main classes using sklearn-decision-tree

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 10.29 | 25.76 | 36.61 | 27.34 |
| Ground | 0.09 | 64.96 | 34.05 | 0.90 |
| Vertical | 0.05 | 14.18 | 83.07 | 2.71 |
| Sky | 0.34 | 3.46 | 22.70 | 73.50 |

4.1.1.2. *Sub Classes.* The baseline confusion matrix and box-whisker plot for the subclasses are shown in Table 4.4 and Figure 4.2 respectively. We can see that the results for the sub-classes are quite different from those of the main-classes. The per-label accuracy is extremely poor, except for POROUS. This implies that the features used do not discriminate very well for most of the sub-class labels. The reason POROUS performs quite well could be due to the fact that porous objects have more texture as compared to solid planar surfaces. For example, a

wire mesh or foliage has more texture than compared to planar surfaces such as walls. So, the texture features in the feature set might be contributing to classifying POROUS.

**TABLE 4.4.** Confusion matrices for sub classes using liblinear

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 84.95 | 0.25 | 2.34 | 0.64 | 7.39 | 4.42 |
| Left | 28.45 | 7.70 | 14.64 | 3.97 | 26.07 | 19.18 |
| Center | 37.95 | 2.67 | 16.61 | 5.91 | 22.67 | 14.19 |
| Right | 25.45 | 2.00 | 16.42 | 14.36 | 26.87 | 14.90 |
| Porous | 17.13 | 0.58 | 1.93 | 1.48 | 73.68 | 5.20 |
| Solid | 39.39 | 1.16 | 6.14 | 2.35 | 20.87 | 30.09 |

**TABLE 4.5.** Confusion matrices for sub classes using sklearn-liblinear

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 86.36 | 0.16 | 2.13 | 0.54 | 6.89 | 3.92 |
| Left | 29.48 | 4.64 | 15.03 | 4.92 | 26.78 | 19.15 |
| Center | 39.45 | 1.23 | 16.34 | 5.29 | 24.00 | 13.69 |
| Right | 27.71 | 0.82 | 15.47 | 13.11 | 29.03 | 13.85 |
| Porous | 14.62 | 0.26 | 1.41 | 1.11 | 77.85 | 4.75 |
| Solid | 38.34 | 0.53 | 4.94 | 2.08 | 23.48 | 30.64 |

**TABLE 4.6.** Confusion matrices for sub classes using sklearn-decision-tree

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 67.05 | 2.75 | 7.22 | 3.69 | 8.39 | 10.89 |
| Left | 16.70 | 15.04 | 20.61 | 10.50 | 16.14 | 21.01 |
| Center | 19.41 | 8.44 | 22.86 | 13.68 | 17.00 | 18.62 |
| Right | 15.14 | 7.28 | 22.11 | 19.35 | 18.20 | 17.92 |
| Porous | 11.89 | 3.70 | 9.02 | 6.15 | 57.59 | 11.64 |
| Solid | 21.97 | 6.86 | 15.01 | 8.79 | 16.34 | 31.03 |

4.1.2. CLASSIFIER COMPARISONS. Here, we substitute the SVM using an alternate implementation of the linear kernel and a decision tree classifier. This will tell us if there is any difference between the two SVM implementations as well as between the SVM and decision tree classifier.

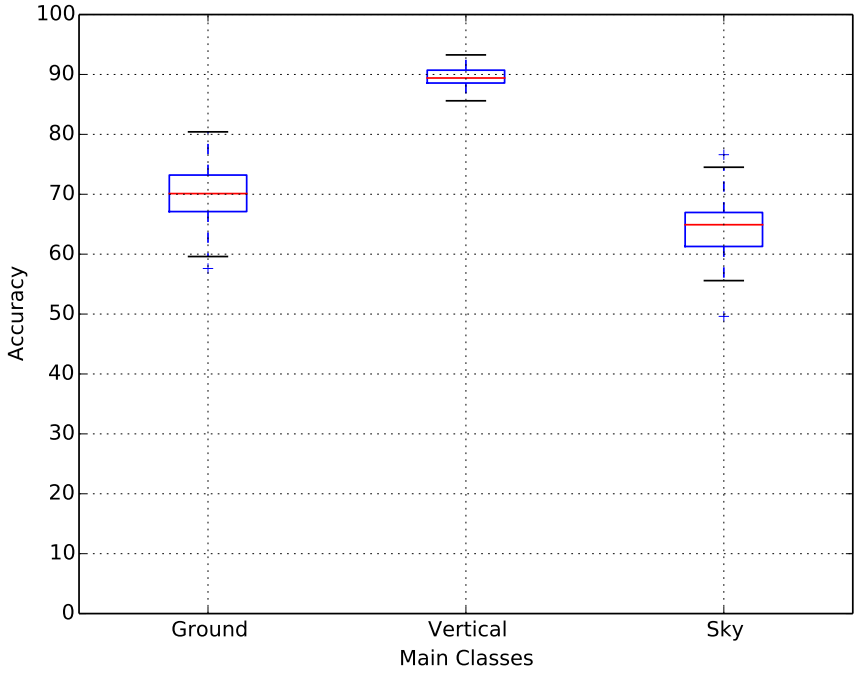



**FIGURE 4.1.** Baseline liblinear boxplot for main classes

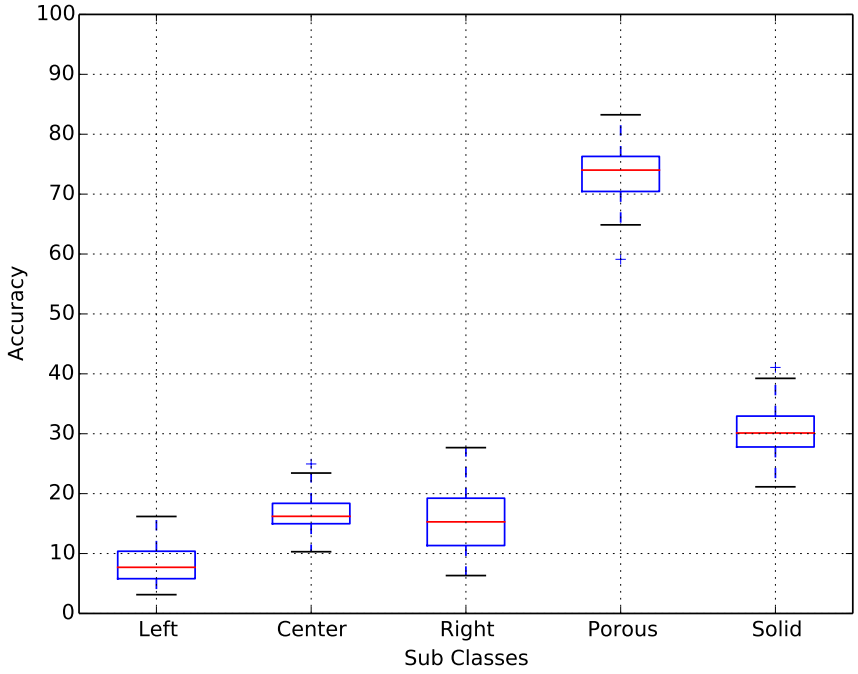
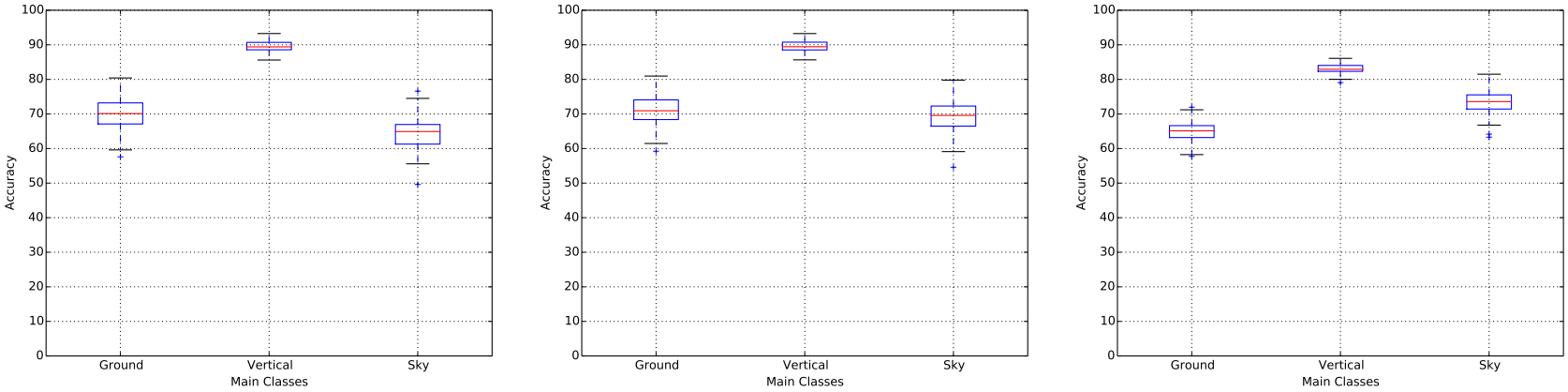



**FIGURE 4.2.** Baseline liblinear boxplot for sub classes

4.1.2.1. *Main Classes.* Figure 4.3 shows the box-plot comparisons for the main class classification. Tables 4.2 and 4.3 shows the corresponding confusion matrix. From these comparisons we can see that liblinear is the more stable classifier as it has a smaller quartile range though sklearn-liblinear has better maximum performance at times. The vanilla decision tree's performance is also shown and this is lower than the other two. As far as classifier selection is concerned, both SVM implementations outperform the decision tree classifier for GROUND and VERTICAL whereas the decision tree classifier gets better results for SKY. We also observe that both GROUND and SKY are mostly misclassified as VERTICAL. This could be due to classification being dependent on location features and the fact that VERTICAL surfaces are in between GROUND and SKY surfaces resulting in some ambiguity in classification. We also perform a test of statistical significance on the difference between the classifiers being compared. While we obtained the difference between our baseline and decision-tree classifier as statistically significant, the difference between baseline and sklearn-liblinear was statistically inconclusive. This is because the p-values from McNemar's test were in the range of 0 and 0.8.

4.1.2.2. *Sub Classes.* Figure 4.4 shows the box-plot comparisons for the subclass classification. Tables 4.6 and 4.5 show the corresponding confusion matrix. For POROUS both SVM implementations give over 70% average accuracy over 20 runs whereas the decision tree classifier only reaches about 57%. This is much more than our evaluation criteria of 5% difference being meaningful. We can thus see that choice of classifier does indeed make a huge difference for both the main and sub classes. For sub-classes other than POROUS the performance is quite poor. This could be due to the features being unable to discriminate between left, center and right facing surfaces as found in the dataset. Moreover, non-planar solid surfaces is confused with porous and center-facing surfaces.

Once again, we perform McNemar's test in order to look for a statistical significance between classifiers while classifying the sub classes. Again we find that the difference between our baseline and decision-tree classifier is statistically significant, but the difference between baseline and sklearn-liblinear was statistically inconclusive. This is because the p-values from McNemar's test were in the range of 0 and 0.07.

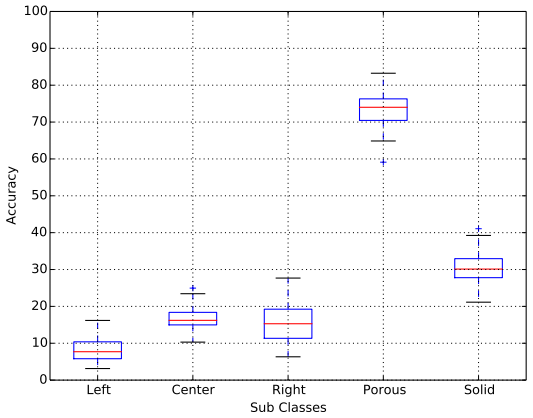



(A) Liblinear (B) Sklearn-liblinear (C) Decision Tree

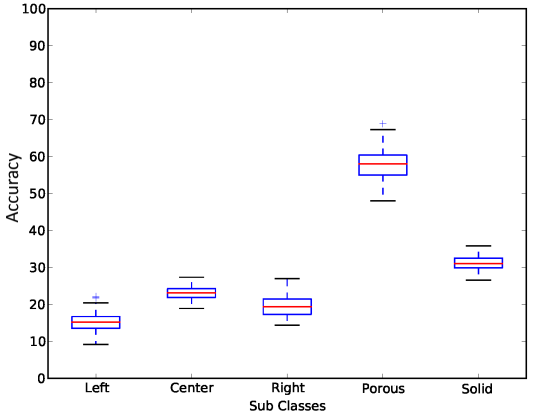
**FIGURE 4.3.** Classifier Comparisons for Main Class

4.1.3. USING NEIGHBORHOOD INFORMATION. In this section, we see the result of changing a superpixel label based on some condition of its neighbors. For this experiment we only consider liblinear as a classifier. The results of this experiment are shown in Figure 4.5. The legend is explained as follows. $C_{baseline}$ is the case where we plot the average accuarcy of the classifier over 50 runs of 5-fold cross validation. $C_{majority}$ is the case where we change the label if a given superpixel is surrounded by a majority of differing labels. $C_{neighbors}$ is the case where we change the label only if the absolute difference between the given label and the majority of its neighbors is 1. $C_{size}$ is the case where we change the label only of the superpixel size is less than 200. And finally $C_{threshold}$ indicates the accuracy after changing the label if the probability of classification of a given superpixel is less than 0.5.

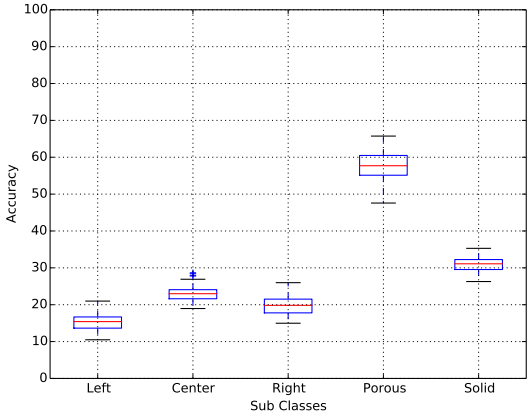
As seen in Figure 4.5 overall we get a marginal improvement in accuracy. This is true for all the conditions where we change the labels of a superpixel. On a per-label basis the accuracy is either around or slightly greater than $C_{baseline}$ for VERTICAL and GROUND. For SKY, however, accuracy reduces drastically in all cases except for $C_{threshold}$. These observations imply that using local neighborhood information is of some use for VERTICAL and GROUND but not for SKY. Moreover, such an analysis makes sense only for the main classes but not so much for the sub-classes. This is due to the fact that, in the case of sub-class labels, a porous object could be surrounded by non-porous objects and likewise for the other labels. Nonetheless, we provide plots for sub-classes for the sake of completion as shown in Figure 4.6. Overall we can surmise that adding local contextual information in this fashion is not as effective as the approach followed by Hoiem et al. [12], who add contextual information more globally.




(A) Liblinear




(B) Sklearn-liblinear




(C) Decision Tree

**FIGURE 4.4.** Classifier Comparisons for Sub Class

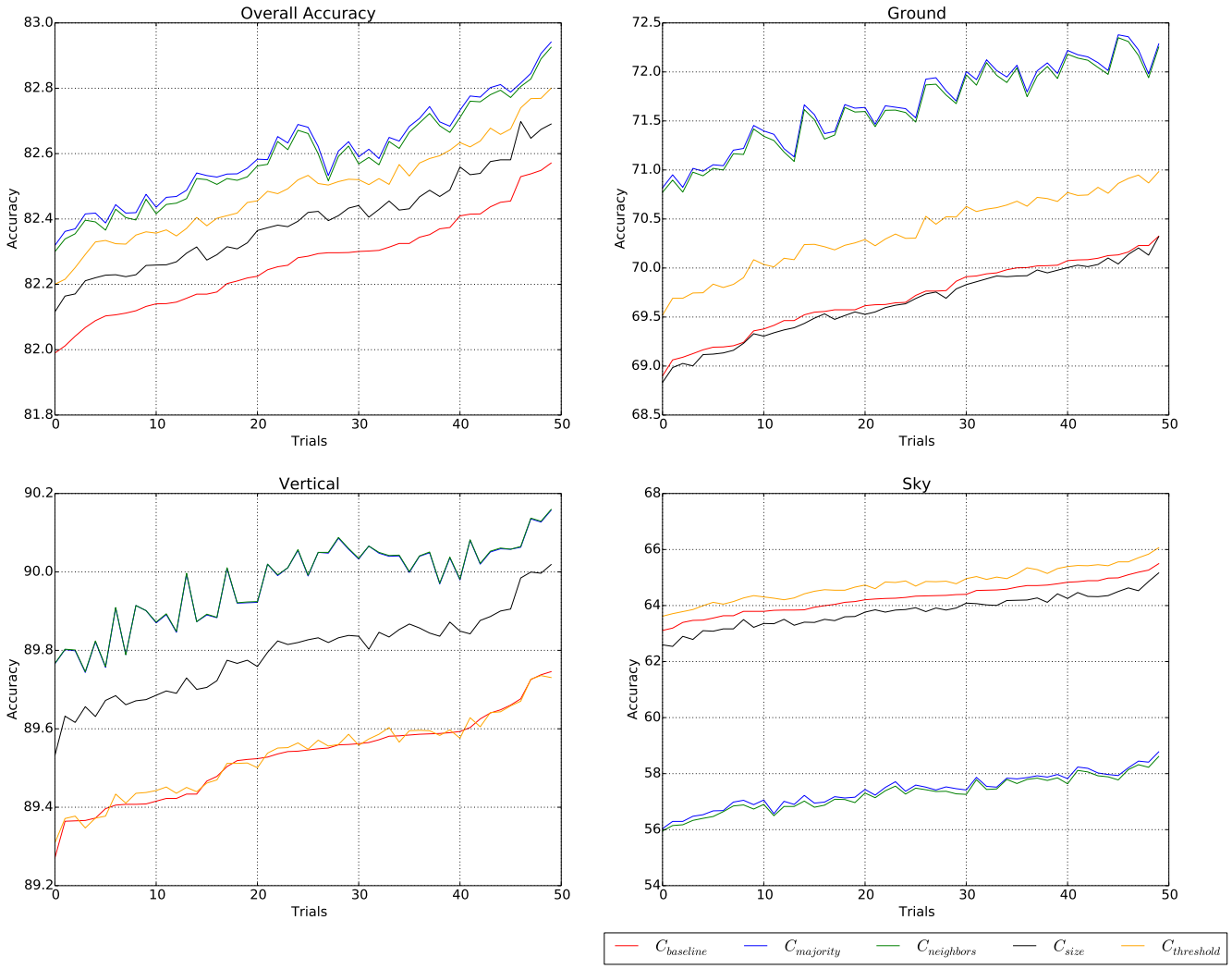



**FIGURE 4.5.** Label Change (Main Class)

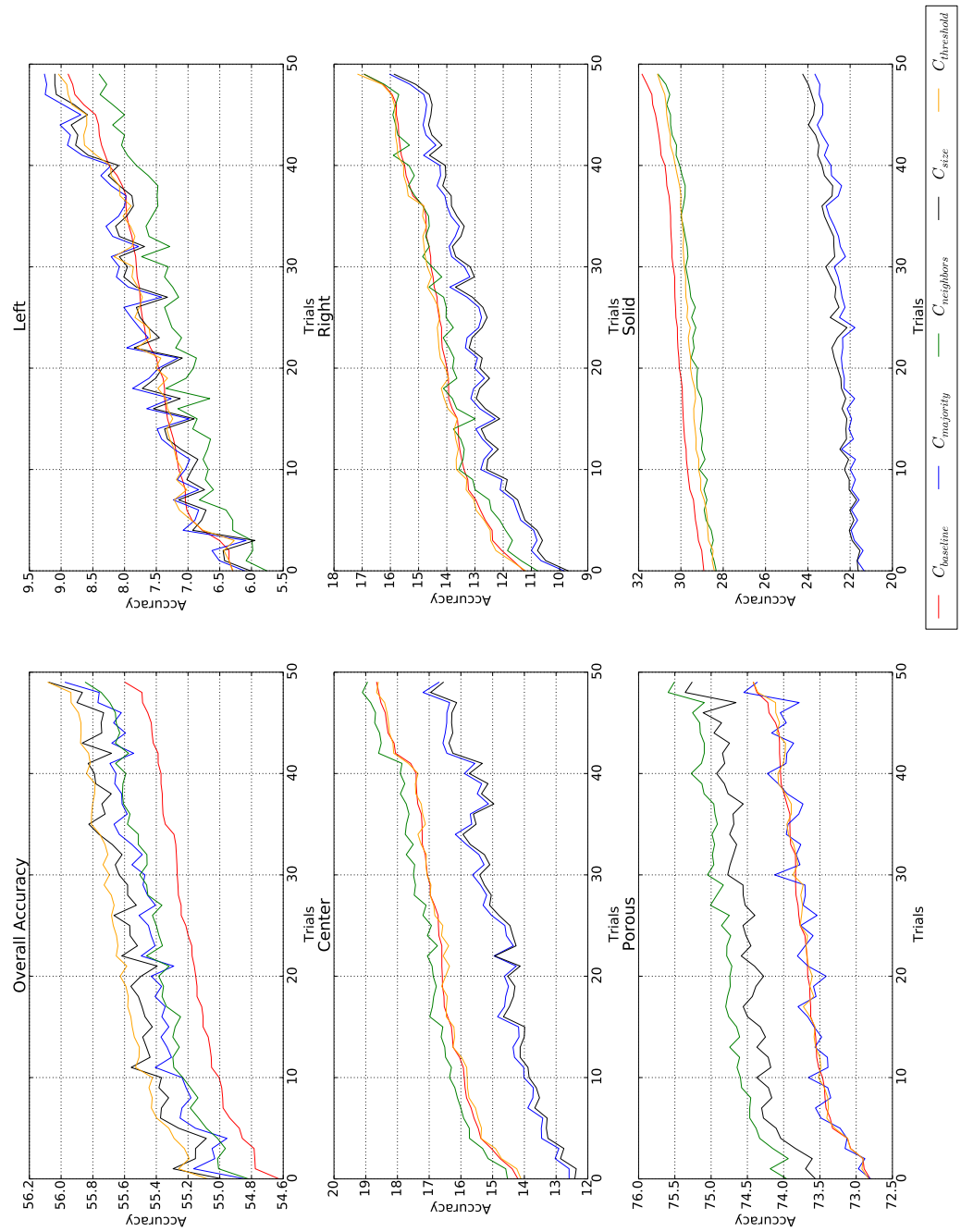



**FIGURE 4.6.** Label Change (Sub Class)

4.1.4. FEATURE SELECTION USING RFE. Recursive Feature Elimination (RFE) allows us to iteratively remove the lowest scoring features such that we are left with only the most important features. The plots in Figures 4.7 and 4.8 show the number of important features obtained after using a varying number of images. For the main class around 10 to 15 features should be more than enough to get a reasonable gain. For the sub class we need a higher number of features. Using all 52 features results in only a marginal increase in performance. We also list the most important features for the main class in Table 4.8a and for the sub class in Table 4.9a. We see that color and position related features rank highly for both main as well as sub classes.

4.1.5. FEATURE SELECTION USING TREES. In this experiment, we use a tree-based feature selection method to get the relevant feature set. As seen in Figure 4.9 where we get the important features for all images considered together, we could consider only the first 10-15 features

**TABLE 4.7.** Complete Feature List

| Feature Number | Feature Description |
|---|---|
| 0,1,2 | Mean R,G,B values |
| 3,4,5 | Mean HSV |
| 6-10 | Histogram bins for Hue |
| 11-13 | Histogram bins for saturation |
| 14-28 | Texture - Mean absolute responses |
| 29-44 | Texture - Hist of max responses |
| 44 | Mean ‘x’ coordinates |
| 45 | Mean ‘y’ coordinates |
| 46 | 10 percentile of ‘x’ coordinates |
| 47 | 90 percentile of ‘x’ coordinates |
| 48 | 10 percentile of ‘y’ coordinates |
| 49 | 90 percentile of ‘y’ coordinates |
| 50 | Height/Width |
| 51 | Superpixel area |

**TABLE 4.8.** Important Features for Main Class

**(A)** RFE

| Feature Number | Feature Description |
|---|---|
| 0 | Mean red value |
| 1 | Mean green value |
| 2 | Mean blue value |
| 5 | 'V' from mean HSV |
| 29 | Histogram texture 1 |
| 32 | Histogram texture 4 |
| 33 | Histogram texture 5 |
| 42 | Histogram texture 14 |
| 43 | Histogram texture 15 |
| 45 | Mean 'y' coordinates |
| 48 | 10 percentile of 'y' coordinates |
| 49 | 90 percentile of 'y' coordinates |

**(B)** Trees

| Feature Number | Feature Description |
|---|---|
| 0 | Mean red value |
| 1 | Mean green value |
| 2 | Mean blue value |
| 5 | 'V' from mean HSV |
| 8 | Hue histogram 3 |
| 17 | Mean texture 4 |
| 23 | Mean texture 10 |
| 24 | Mean texture 11 |
| 32 | Histogram texture 4 |
| 45 | Mean 'y' coordinates |
| 48 | 10 percentile of 'y' coordinates |
| 49 | 90 percentile of 'y' coordinates |

**TABLE 4.9.** Important Features for Sub Class

**(A)** RFE

| Feature Number | Feature Description |
|---|---|
| 0 | Mean red value |
| 1 | Mean green value |
| 2 | Mean blue value |
| 3 | ‘H’ in mean HSV |
| 4 | ‘S’ in mean HSV |
| 5 | ‘V’ from mean HSV |
| 6 | Histogram Hue 1 |
| 8 | Histogram Hue 3 |
| 9 | Histogram Hue 4 |
| 10 | Histogram Hue 5 |
| 11 | Histogram saturation 1 |
| 12 | Histogram saturation 2 |
| 13 | Histogram saturation 3 |
| 29 | Histogram texture 1 |
| 30 | Histogram texture 2 |
| 32 | Histogram texture 4 |
| 33 | Histogram texture 5 |
| 35 | Histogram texture 7 |
| 38 | Histogram texture 10 |
| 39 | Histogram texture 11 |
| 41 | Histogram texture 13 |
| 42 | Histogram texture 14 |
| 43 | Histogram texture 15 |
| 44 | Mean ‘x’ coordinates |
| 45 | Mean ‘y’ coordinates |
| 46 | 10 percentile of ‘x’ coordinates |
| 47 | 90 percentile of ‘x’ coordinates |
| 48 | 10 percentile of ‘y’ coordinates |
| 49 | 90 percentile of ‘y’ coordinates |

**(B)** Trees

| Feature Number | Feature Description |
|---|---|
| 0 | Mean red value |
| 1 | Mean green value |
| 2 | Mean blue value |
| 3 | ‘H’ in mean HSV |
| 5 | ‘V’ from mean HSV |
| 6 | Hue histogram bin 1 |
| 7 | Hue histogram bin 2 |
| 11 | Saturation histogram bin 1 |
| 12 | Saturation histogram bin 2 |
| 17 | Mean texture 4 |
| 22 | Mean texture 9 |
| 23 | Mean texture 10 |
| 24 | Mean texture 11 |
| 32 | Histogram texture 4 |
| 42 | Histogram texture 14 |
| 45 | Mean ‘y’ coordinates |
| 48 | 10 percentile of ‘y’ coordinates |
| 49 | 90 percentile of ‘y’ coordinates |

as being important. Similar figures for images numbering 100 and 200 are shown in Appendix C. If we take the first few features common to each set from each of the graphs in Appendix C, we get the list of features mentioned in Table 4.8b. Likewise, Table 4.9b shows the most important features for the sub-classes obtained using this method. We see once again, that color and position related features rank highly for both main as well as sub classes.

4.1.6. CLASSIFICATION USING REDUCED SET OF FEATURES. In this section, we use the reduced feature set in order to train an SVM classifier. The results are seen in Figures 4.11, 4.12, 4.13 and 4.14. We can see that for most of the labels in the main and sub class categories the performance is very close to that of the full feature set. This is true for reduced features obtained using both the tree-based method and the RFE method.

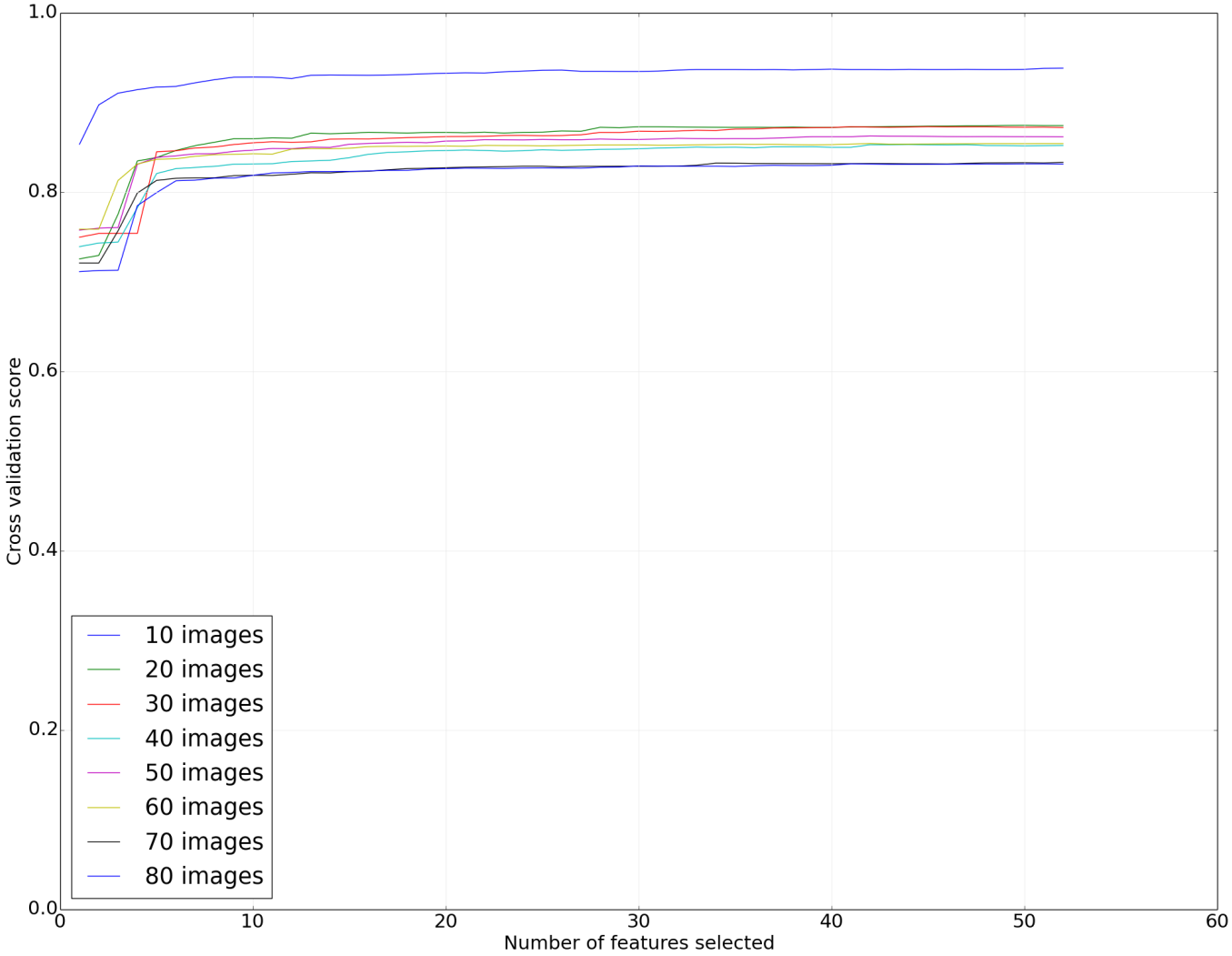



**FIGURE 4.7.** Recursive Feature Elimination (Main Class)

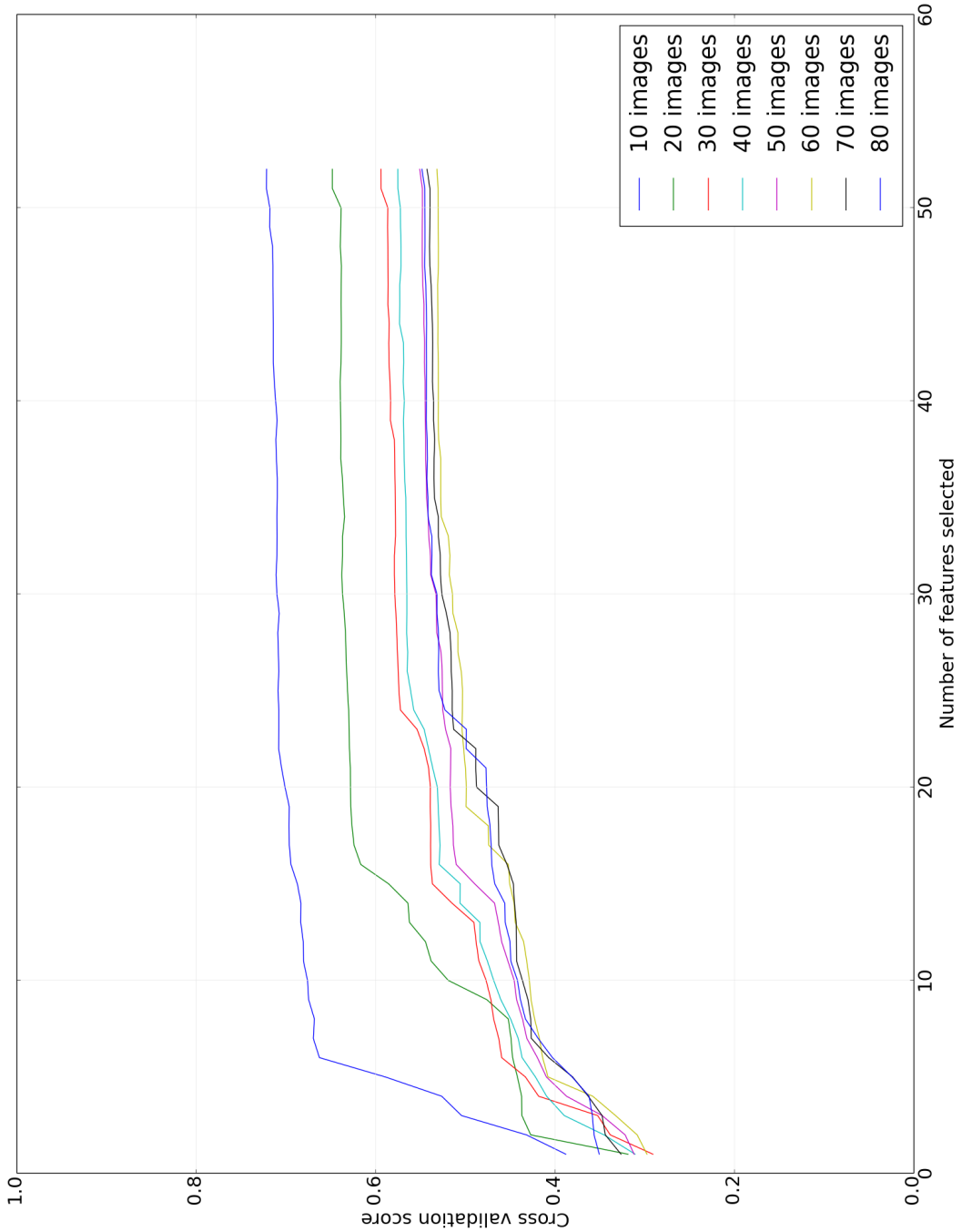



**FIGURE 4.8.** Recursive Feature Elimination (Sub Class)

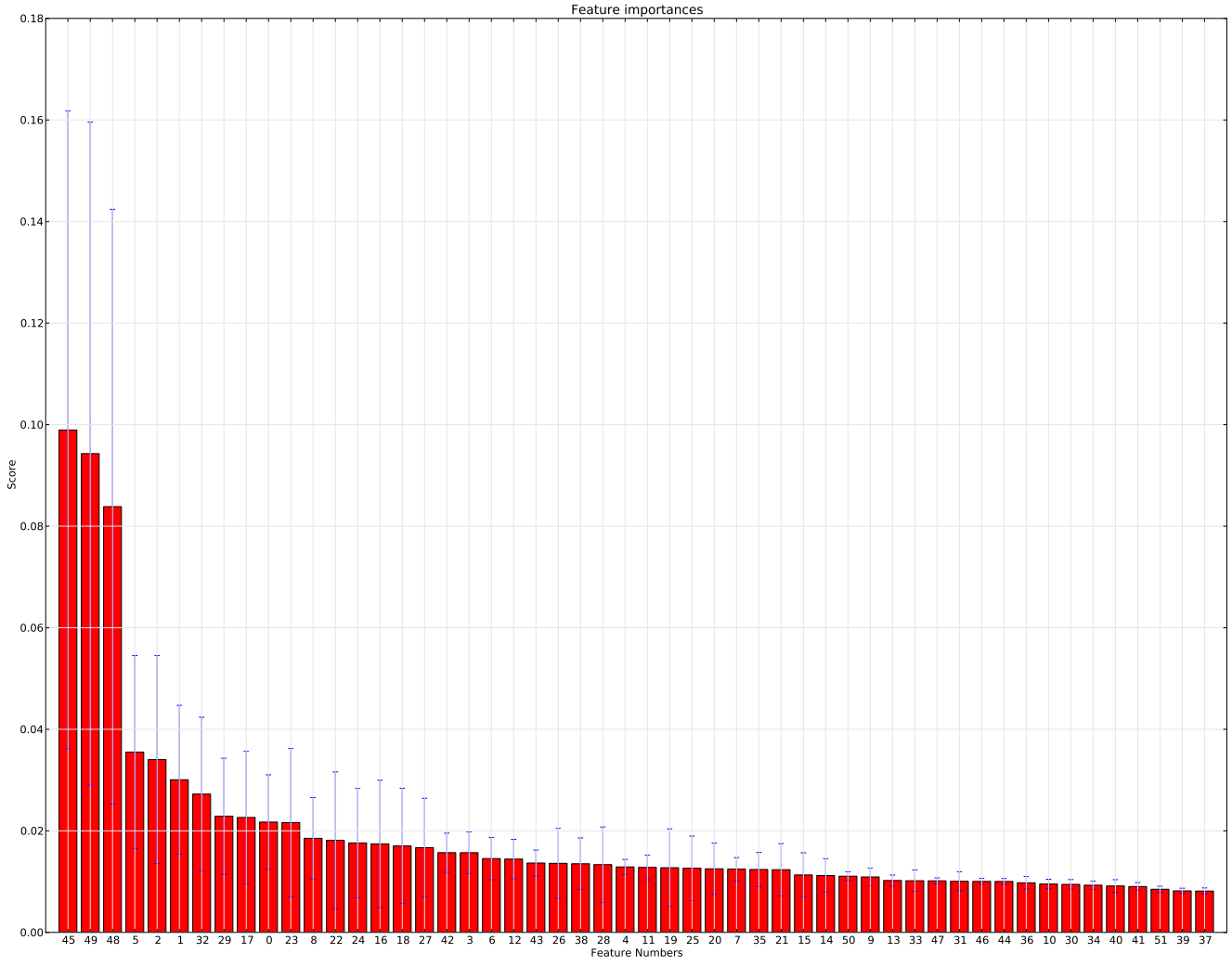



**FIGURE 4.9.** Tree-based Feature Selection for main class using all 300 images

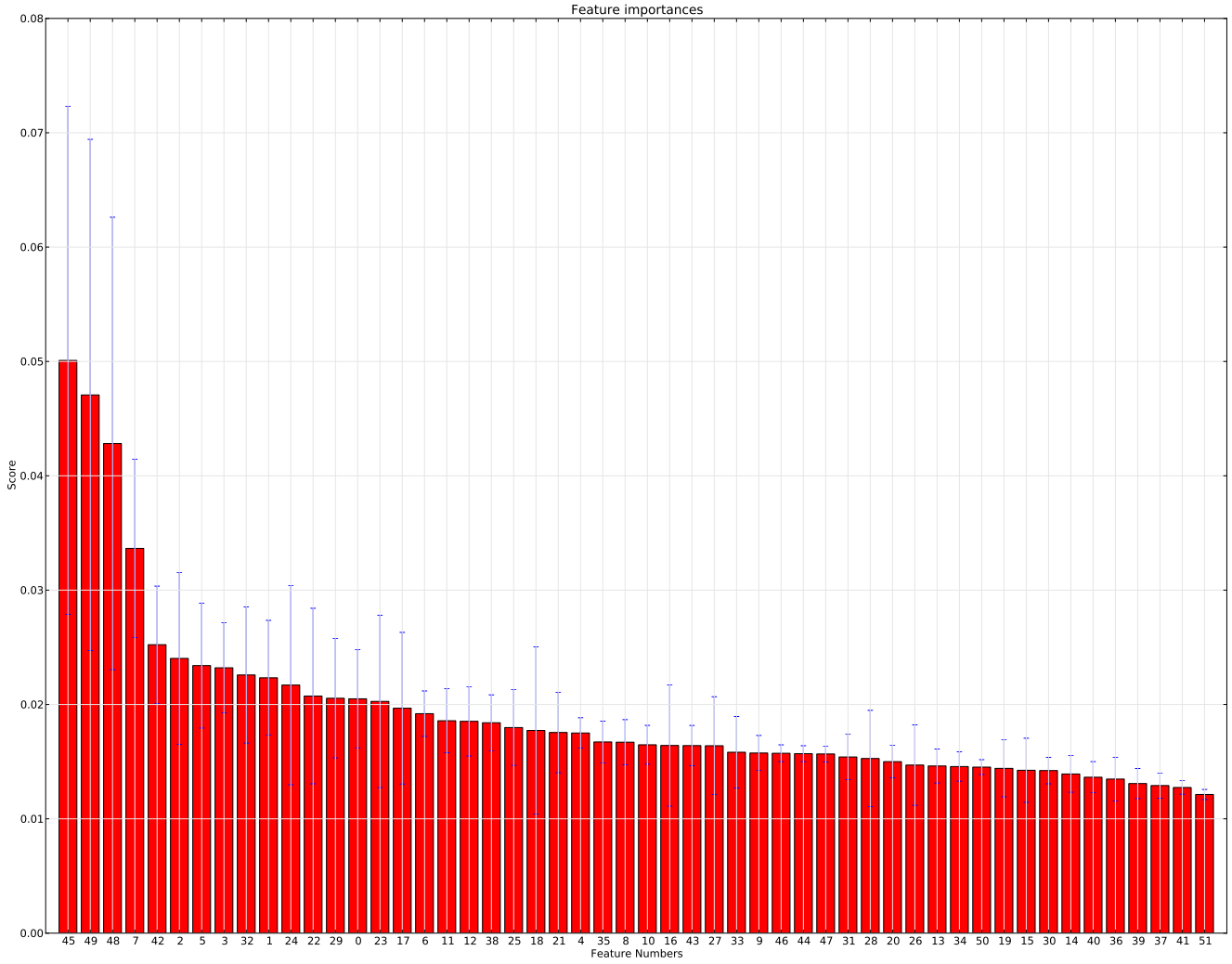



**FIGURE 4.10.** Tree-based Feature Selection for sub class using all 300 images

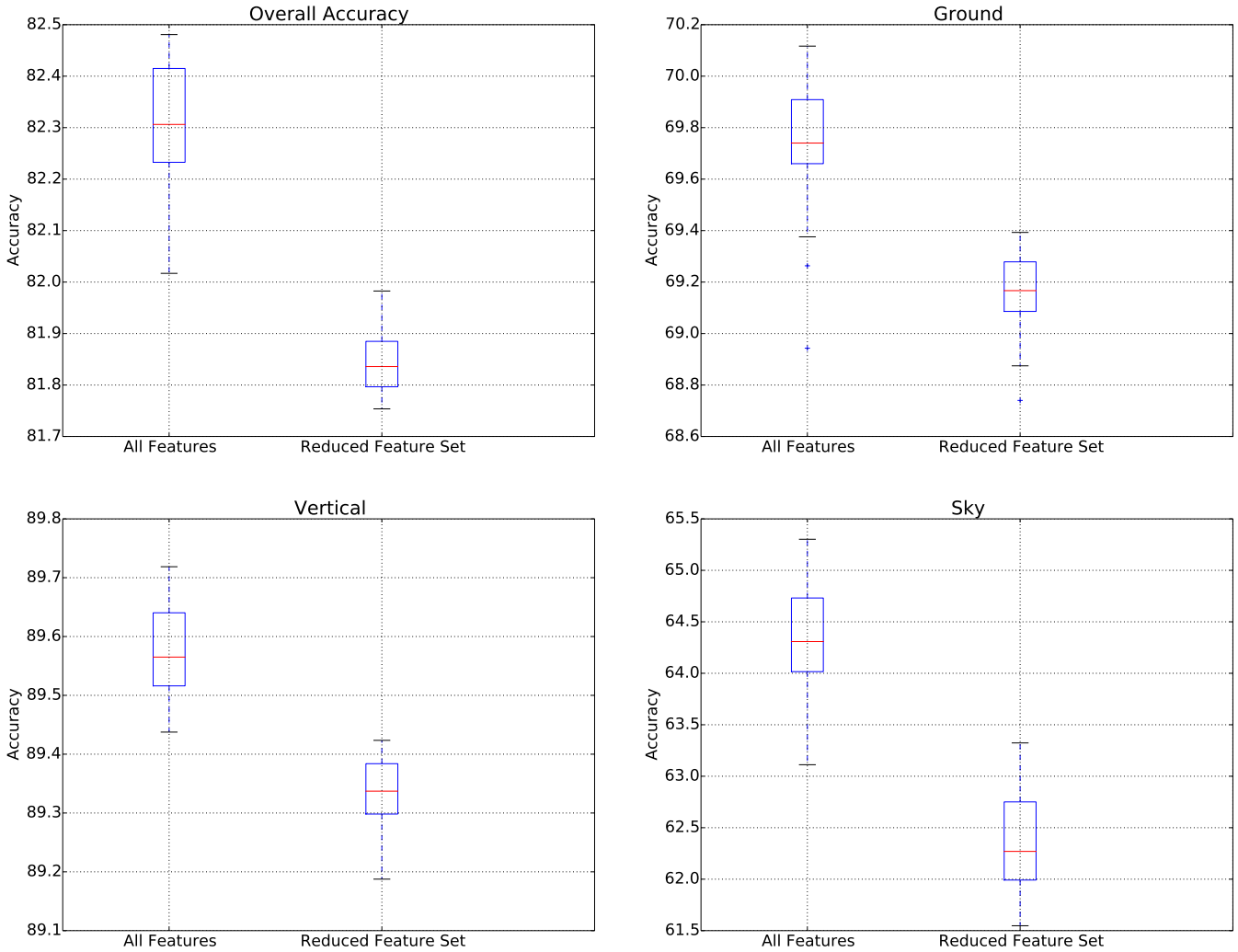



**FIGURE 4.11.** Classification using Reduced Features from Tree-based Selection (Main class)

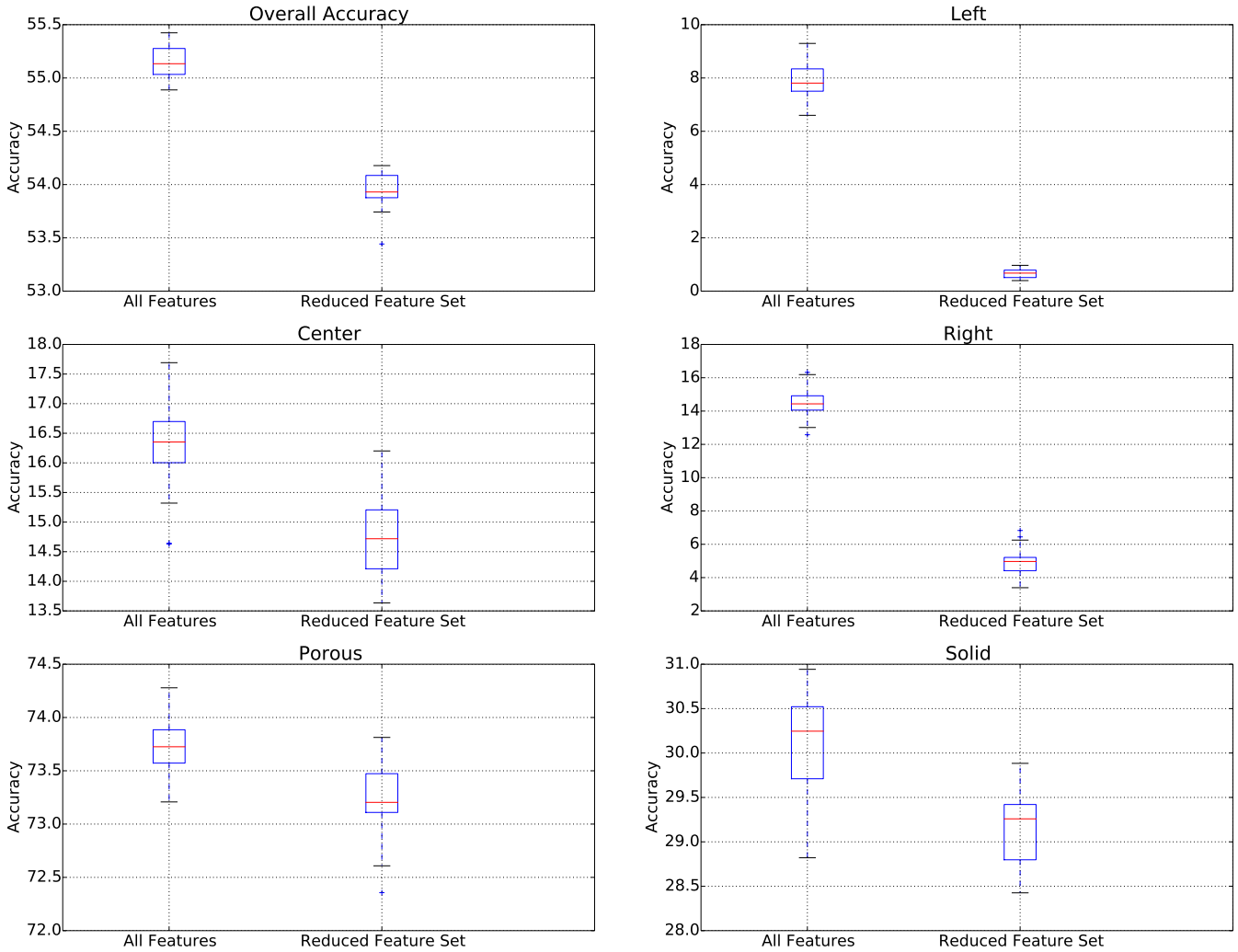



**FIGURE 4.12.** Classification using Reduced Features from Tree-based Selection (Sub class)

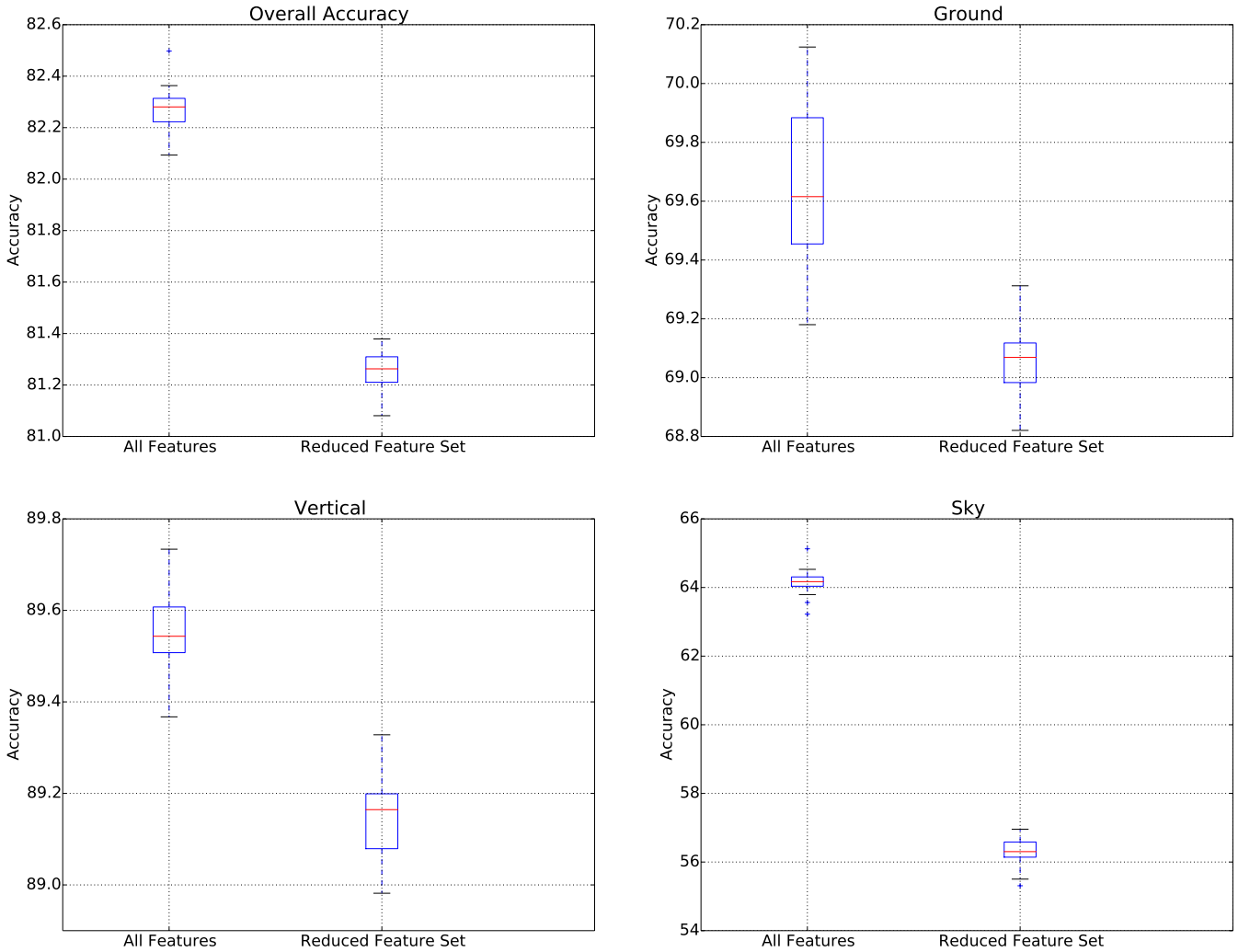



**FIGURE 4.13.** Classification using Reduced Features from RFE (Main class)

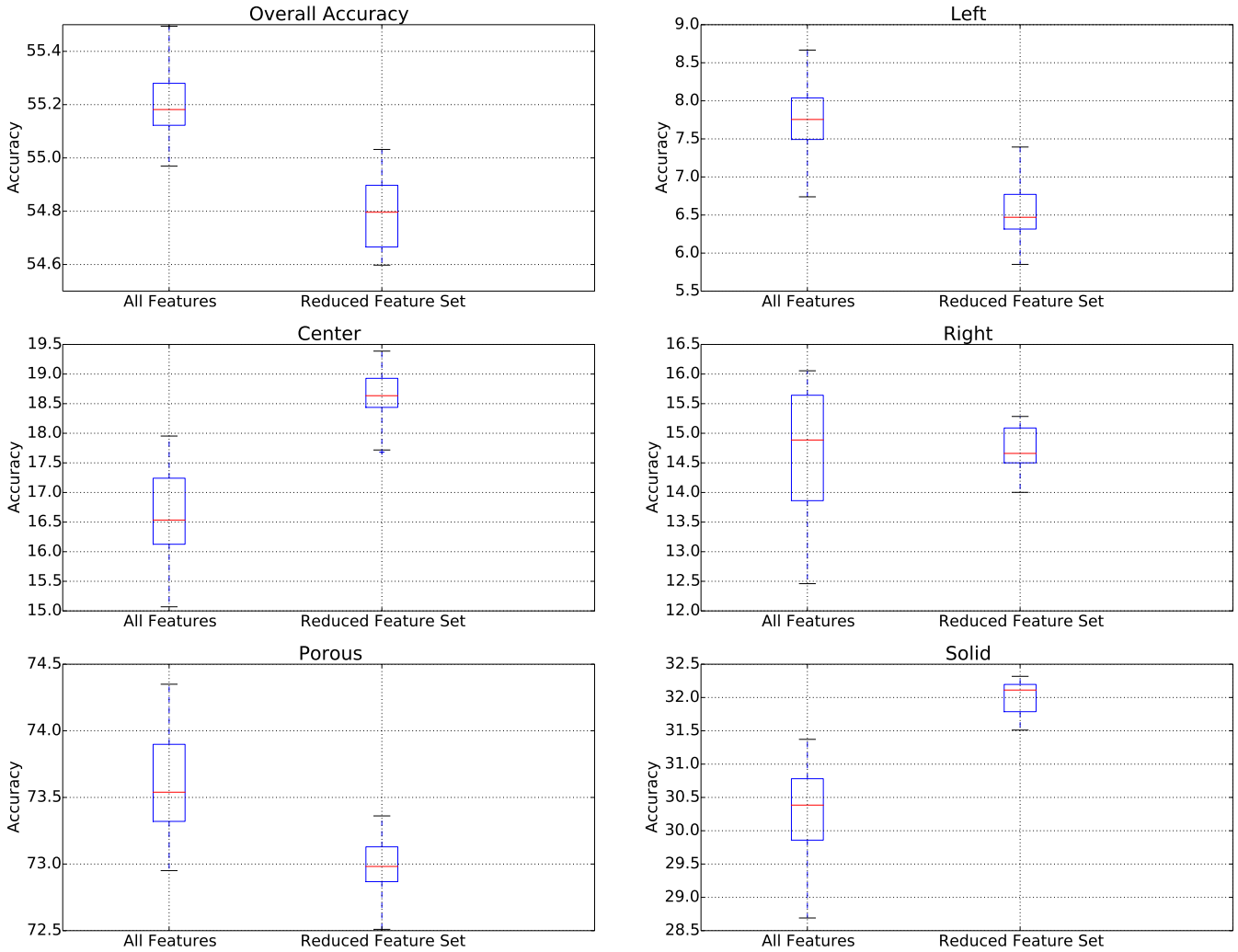



**FIGURE 4.14.** Classification using Reduced Features from RFE (Sub class)

CHAPTER 5

# CONCLUSION AND FUTURE WORK

In this chapter we summarize our results and draw some conclusions. Feature selection showed that color and location-based features are the most important for main-class identification. In addition to these we need texture-based features to identify the sub-classes. Feature selection in our experiments resulted in only a marginal decrease in labeling accuracy. It was also observed that the time it takes to train the classifier is drastically reduced when we use only the most important features. This could be useful for certain kinds of applications, like those running on embedded devices with constrained computational support. While this may not be real-time, it surely is one step closer.

The average accuracy that we obtained in the absence of contextual information was 82% for the main classes and around 55% for the sub classes. Compare this to the numbers obtained by Hoiem et al. [12], which come up to 88% and 61% for the main and sub classes respectively. Hence, we can conclude that, using the additional information provided by context via the multi-segmentation approach does in fact improve labeling accuracy. However, the fact that the accuracy for the main classes is as high as 82% in our case and 86% in Hoiem's case suggests that the features used were quite discriminative in the first place. Moreover, as mentioned previously, color and location information were the most important for classifying the main classes. This, however, was not the case for the sub-classes. Except for POROUS, subclass accuarcy was relatively poor and it only got worse using the reduced feature set. This means that context really helped in labeling the sub classes.

Using local neighborhood information, while possibly a sensible approach only for the main classes, resulted in about a percentage increase on average. Classification of ground and

vertical classes improved in all cases, while labeling sky using this technique reduced accuracy in most of the cases. Clearly, adding contextual information using the multiple-segmentation procedure improved labeling accuracy. This means that we need to add contextual information at a global level instead of at a local level as we have done.

As for the classifiers, the support vector machine implementation of liblinear turned out to be more stable than the rest, while being quite fast in training and testing. Though the reported performance for liblinear is lower than those reported in Hoiem et al. [12] using Matlab's boosted decision tree classifier, this may be an implementation issue. This fact needs to be explored further.

In future work, there is a need to explore a better set of features to supplement the reduced feature set. If we can find newer features with more discriminative power, then the accuracy of using just the superpixel features could improve. This, in turn, could have a cascading effect in the multi-segmentation stage as well, resulting in better per-class labeling accuracy.

We have laid the framework for evaluating a few of the classifiers. This could be extended to include some more classifiers and evaluate which ones perform better. Also, in comparing classifiers we used the McNemar's test to evaluate the statistical difference in performance between classifiers. McNemar's test is designed to work in a scenario where we sample a population only once. This has been a restriction in our case due to having just 300 images to sample from repeatedly. So, in the future we could either run our experiments on more datasets or use a more robust test measure that caters to our case.

While we observed that using local neighborhood information did not help much, we could explore other means of adding contextual information to the system. This would have to be done in a much larger neighborhood than what we have considered at the moment.

# BIBLIOGRAPHY


[1] Radhakrishna Achanta, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels. 12

[2] Claude R. Brice and Claude L. Fennema. Scene analysis using regions. *Artificial Intelligence*, 1(3-4):205–226, 1970. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(70)90008-1. URL http://www.sciencedirect.com/science/article/pii/0004370270900081. 5

[3] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. 20

[4] S.K. Divvala, D. Hoiem, J.H. Hays, A.A. Efros, and M. Hebert. An empirical study of context in object detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1271–1278, June 2009. doi: 10.1109/CVPR.2009.5206532. 9

[5] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. 19

[6] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004. 10, 12, 13

[7] Stephen Gould, Richard Fulton, and Daphne Koller. Decomposing a scene into geometric and semantically consistent regions. In *ICCV*, 2009. 6

[8] Adolfo Guzman-Arenas. Computer recognition of three-dimensional objects in a visual scene. 1968. 5

[9] A. R. Hanson and E. M. Riseman. VISIONS: A computer system for interpreting scenes. In A. R. Hanson and E. M. Riseman, editors, *Computer Vision Systems*. Academic Press, New York. 5

[10] Derek Hoiem, Alexei A Efros, and Martial Hebert. Automatic photo pop-up. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 577–584. ACM, 2005. 1, 2, 7, 8

[11] Derek Hoiem, Alexei A Efros, and Martial Hebert. Geometric context from a single image. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 654–661. IEEE, 2005. 1, 2, 7, 12, 15, 18, 52

[12] Derek Hoiem, Alexei A Efros, and Martial Hebert. Recovering surface layout from an image. *International Journal of Computer Vision*, 75:151–172, 2007. 1, 2, 3, 4, 5, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 31, 47, 48, 52, 53

[13] Thomas Leung and Jitendra Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001. 16

[14] S.E. Palmer. *Vision Science: Photons to Phenomenology*. A Bradford book. BRADFORD BOOK, 1999. ISBN 9780262161831. URL `http://books.google.com/books?id=IEl4QgAACAAJ`. 14

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. 20

[16] L.G. Roberts. *Machine Perception of the Three-dimensional Solids*. Its Technical report. MIT Document Services, 1963. URL `http://books.google.com/books?id=HS97GwAACAAJ`. 5

[17] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. 3dd depth reconstruction from a single still image. *Int. J. Comput. Vision*, 76(1):53–69, January 2008. ISSN 0920-5691. doi: 10.1007/s11263-007-0071-y. URL http://dx.doi.org/10.1007/s11263-007-0071-y. 6, 7

[18] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3d: Learning 3d scene structure from a single still image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31 (5):824–840, 2009. 6

[19] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, Aug 2000. ISSN 0162-8828. doi: 10.1109/34.868688. 12

APPENDIX A

# MISCELLANEOUS

**TABLE A.1.** Superpixel features (C1-C2, T1-T7, L7) and constellation features (all)[11]

| | |
|---|---|
| **Location/Shape** | |
| L1 | Location: normalized x and y, mean |
| L2 | Location: normalized x and y, 10th and 90th percentile |
| L3 | Location: normalized y wrt estimated horizon, 10th and 90th percentile |
| L4 | Shape: number of superpixels in constellation |
| L5 | Shape: number of sides of convex hull |
| L6 | Shape: num of pixels/area(convex hull) |
| L7 | Shape: whether the constellation region is contiguous |
| **Color** | |
| C1 | RGB values: mean |
| C2 | HSV values: conversion from mean rgb values |
| C3 | Hue: histogram (5 bins) and entropy |
| C4 | Saturation: histogram (3 bins) and entropy |
| **Texture** | |
| T1 | DOOG Filters: mean abs response |
| T2 | DOOG Filters: mean of variables in T1 |
| T3 | DOOG Filters: id of max of variables in T1 |
| T4 | DOOG Filters: (max - median) of variables in T1 |
| T5 | Textons: mean abs response |
| T6 | Textons: max of variables in T5 |
| T7 | Textons: (max - median) of variables in T5 |
| **3D Geometry** | |
| G1 | Long Lines: total number in constellation |
| G2 | Long Lines: % of nearly parallel pairs of lines |
| G3 | Line Intersection: histogram over 12 orientations, entropy |
| G4 | Line Intersection: % right of center |
| G5 | Line Intersection: % above center |
| G6 | Line Intersection: % far from center at 8 orientations |
| G7 | Line Intersection: % very far from center at 8 orientations |
| G8 | texture gradient: x and y "edginess" (T2) center |

In [11], Hoiem et al. refered to groups of superpixels used in multiple segmentations as *constellations*. This terminology changed in [12] from constellations to *segments* as seen in Table A.2.

[1]Vanishing Points

**TABLE A.2.** Superpixel features (L1,L6,C1,C2,C3,C4,T1,T2) and segment features (all)[12]

| | |
|---|---|
| **Location/Shape** | |
| L1 | Location: normalized x and y, mean |
| L2 | Location: normalized x and y, 10th and 90th percentile |
| L3 | Location: normalized y wrt estimated horizon, 10th and 90th percentile |
| L4 | Location: whether segment is above, below or straddles est. horizon |
| L5 | Shape: number of superpixels in segment |
| L6 | Shape: normalized area in image |
| **Color** | |
| C1 | RGB values: mean |
| C1 | HSV values: C1 in HSV space |
| C3 | Hue: histogram (5 bins) |
| C4 | Saturation: histogram (3 bins) |
| **Texture** | |
| T1 | LM filters: mean absolute response (15 filters) |
| T2 | LM filters: histogram of maximum responses (15 bins) |
| **Perspective** | |
| P1 | Long Lines: (number of line pixels)/sqrt(area) |
| P2 | Long Lines: percent of nearly parallel pairs of lines |
| P3 | Line Intersections: histogram over 8 orientations, entropy |
| P4 | Line Intersections: percent right of image center |
| P5 | Line Intersections: percent above image center |
| P6 | Line Intersections: percent far from image center at 8 orientations |
| P7 | Line Intersections: percent very far from image center at 8 orientations |
| P8 | VP[1] (num line pixels with vertical VP membership)/sqrt(area) |
| P9 | VP: (num line pixels with horizontal VP membership)/sqrt(area) |
| P10 | VP: percent of total line pixels with vertical VP membership |
| P11 | VP: x-pos of horizontal VP - segment center (0 if none) |
| P12 | VP: y-pos of highest/lowest vertical VP wrt segment center |
| P13 | VP: segment bounds wrt horizontal VP |
| P14 | Gradient: x, y center of mass of gradient magnitude wrt segment center |

APPENDIX B

# DETAILED RESULTS

**TABLE B.1.** Confusion matrices for main classes using liblinear

**(A)** Original confusion matrix

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 4.93 | 10.27 | 14.19 | 9.01 |
| Ground | 1.76 | 5695.59 | 2429.52 | 52.93 |
| Vertical | 0.28 | 1804.01 | 17836.76 | 282.35 |
| Sky | 1.12 | 55.79 | 718.51 | 1398.18 |

**(B)** Row-normalized confusion matrix

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 12.84 | 26.74 | 36.95 | 23.46 |
| Ground | 0.02 | 69.63 | 29.70 | 0.65 |
| Vertical | 0.00 | 9.05 | 89.53 | 1.42 |
| Sky | 0.05 | 2.57 | 33.06 | 64.33 |

**TABLE B.2.** Confusion matrices for main classes using sklearn-liblinear

**(A)** Original confusion matrix

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 3.03 | 8.50 | 13.36 | 13.51 |
| Ground | 4.03 | 5774.01 | 2321.98 | 79.78 |
| Vertical | 2.95 | 1783.47 | 17840.14 | 296.84 |
| Sky | 2.97 | 54.63 | 601.14 | 1514.86 |

**(B)** Row-normalized confusion matrix

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 7.89 | 22.14 | 34.79 | 35.18 |
| Ground | 0.05 | 70.59 | 28.39 | 0.98 |
| Vertical | 0.01 | 8.95 | 89.54 | 1.49 |
| Sky | 0.14 | 2.51 | 27.66 | 69.69 |

**TABLE B.3.** Confusion matrices for main classes using sklearn-decision-tree

**(A)** Original confusion matrix

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 3.95 | 9.89 | 14.06 | 10.50 |
| Ground | 7.75 | 5313.57 | 2785.04 | 73.44 |
| Vertical | 9.92 | 2824.78 | 16549.76 | 538.94 |
| Sky | 7.41 | 75.13 | 493.51 | 1597.55 |

**(B)** Row-normalized confusion matrix

| | Unlabeled | Ground | Vertical | Sky |
|---|---|---|---|---|
| Unlabeled | 10.29 | 25.76 | 36.61 | 27.34 |
| Ground | 0.09 | 64.96 | 34.05 | 0.90 |
| Vertical | 0.05 | 14.18 | 83.07 | 2.71 |
| Sky | 0.34 | 3.46 | 22.70 | 73.50 |

**TABLE B.4.** Confusion matrices for sub classes using liblinear

**(A)** Original confusion matrix

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 8832.97 | 26.51 | 243.54 | 66.67 | 768.13 | 459.98 |
| Left | 487.17 | 131.84 | 250.70 | 67.97 | 446.34 | 328.38 |
| Center | 1481.83 | 104.39 | 648.42 | 230.78 | 885.31 | 553.87 |
| Right | 607.33 | 47.71 | 391.99 | 342.84 | 641.39 | 355.54 |
| Porous | 1251.17 | 42.37 | 140.88 | 108.26 | 5381.58 | 380.14 |
| Solid | 1815.72 | 53.38 | 282.83 | 108.32 | 962.15 | 1386.80 |

**(B)** Row-normalized confusion matrix

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 84.95 | 0.25 | 2.34 | 0.64 | 7.39 | 4.42 |
| Left | 28.45 | 7.70 | 14.64 | 3.97 | 26.07 | 19.18 |
| Center | 37.95 | 2.67 | 16.61 | 5.91 | 22.67 | 14.19 |
| Right | 25.45 | 2.00 | 16.42 | 14.36 | 26.87 | 14.90 |
| Porous | 17.13 | 0.58 | 1.93 | 1.48 | 73.68 | 5.20 |
| Solid | 39.39 | 1.16 | 6.14 | 2.35 | 20.87 | 30.09 |

**TABLE B.5.** Confusion matrices for sub classes using sklearn-liblinear

**(A)** Original confusion matrix

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 8979.31 | 17.11 | 221.60 | 56.44 | 716.15 | 407.19 |
| Left | 504.83 | 79.51 | 257.33 | 84.26 | 458.59 | 327.88 |
| Center | 1540.24 | 48.02 | 638.12 | 206.63 | 937.21 | 534.38 |
| Right | 661.44 | 19.47 | 369.35 | 312.98 | 692.91 | 330.65 |
| Porous | 1067.88 | 18.78 | 103.12 | 81.00 | 5686.82 | 346.80 |
| Solid | 1766.94 | 24.37 | 227.72 | 95.86 | 1082.05 | 1412.26 |

**(B)** Row-normalized confusion matrix

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 86.36 | 0.16 | 2.13 | 0.54 | 6.89 | 3.92 |
| Left | 29.48 | 4.64 | 15.03 | 4.92 | 26.78 | 19.15 |
| Center | 39.45 | 1.23 | 16.34 | 5.29 | 24.00 | 13.69 |
| Right | 27.71 | 0.82 | 15.47 | 13.11 | 29.03 | 13.85 |
| Porous | 14.62 | 0.26 | 1.41 | 1.11 | 77.85 | 4.75 |
| Solid | 38.34 | 0.53 | 4.94 | 2.08 | 23.48 | 30.64 |

**TABLE B.6.** Confusion matrices for sub classes using sklearn-decision-tree

**(A)** Original confusion matrix

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 6972.05 | 285.98 | 750.83 | 383.63 | 872.62 | 1132.69 |
| Left | 286.01 | 257.48 | 352.92 | 179.79 | 276.36 | 359.84 |
| Center | 757.91 | 329.63 | 892.45 | 533.98 | 663.76 | 726.87 |
| Right | 361.31 | 173.76 | 527.62 | 461.86 | 434.45 | 427.80 |
| Porous | 868.66 | 270.11 | 658.88 | 449.57 | 4206.68 | 850.50 |
| Solid | 1012.81 | 316.30 | 691.81 | 405.21 | 752.93 | 1430.14 |

**(B)** Row-normalized confusion matrix

| | Unlabeled | Left | Center | Right | Porous | Solid |
|---|---|---|---|---|---|---|
| Unlabeled | 67.05 | 2.75 | 7.22 | 3.69 | 8.39 | 10.89 |
| Left | 16.70 | 15.04 | 20.61 | 10.50 | 16.14 | 21.01 |
| Center | 19.41 | 8.44 | 22.86 | 13.68 | 17.00 | 18.62 |
| Right | 15.14 | 7.28 | 22.11 | 19.35 | 18.20 | 17.92 |
| Porous | 11.89 | 3.70 | 9.02 | 6.15 | 57.59 | 11.64 |
| Solid | 21.97 | 6.86 | 15.01 | 8.79 | 16.34 | 31.03 |

APPENDIX C

# TREE-BASED FEATURE SELECTION

The plots on the following pages show the important features for the main and sub classes using the Forest of Trees method of feature selection.

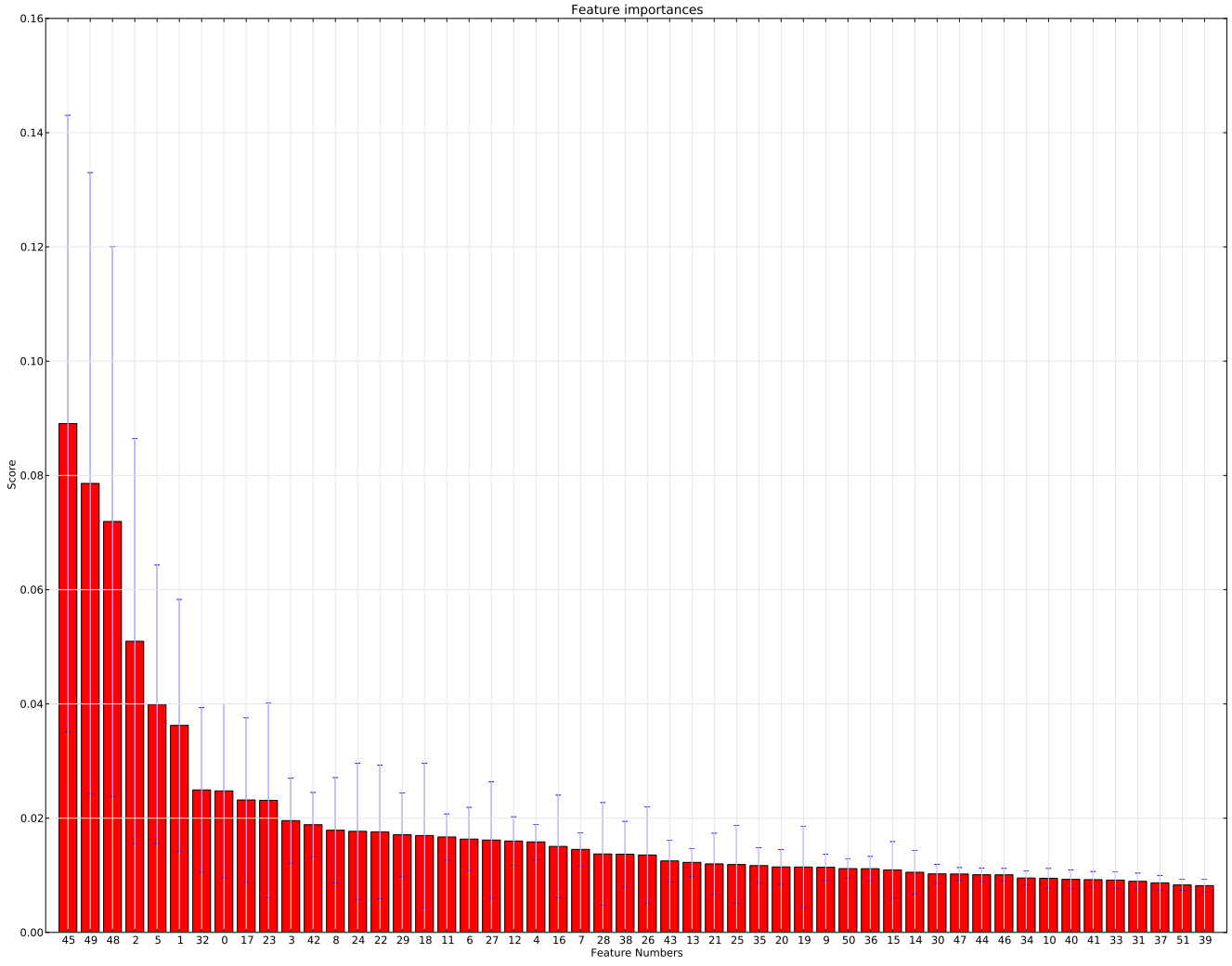



**FIGURE C.1.** Tree-based Feature Selection for main class using 100 images

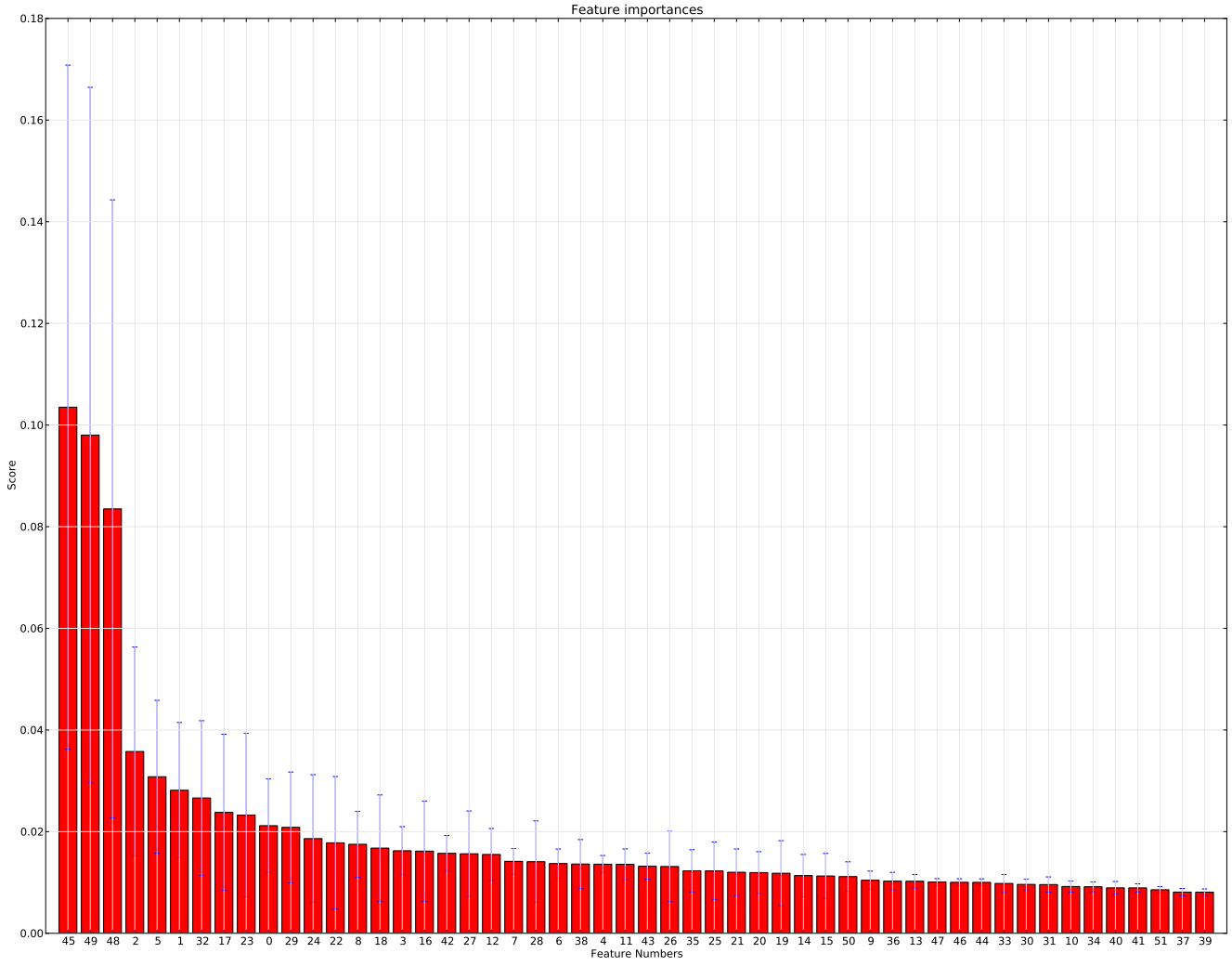



**FIGURE C.2.** Tree-based Feature Selection for main class using 200 images

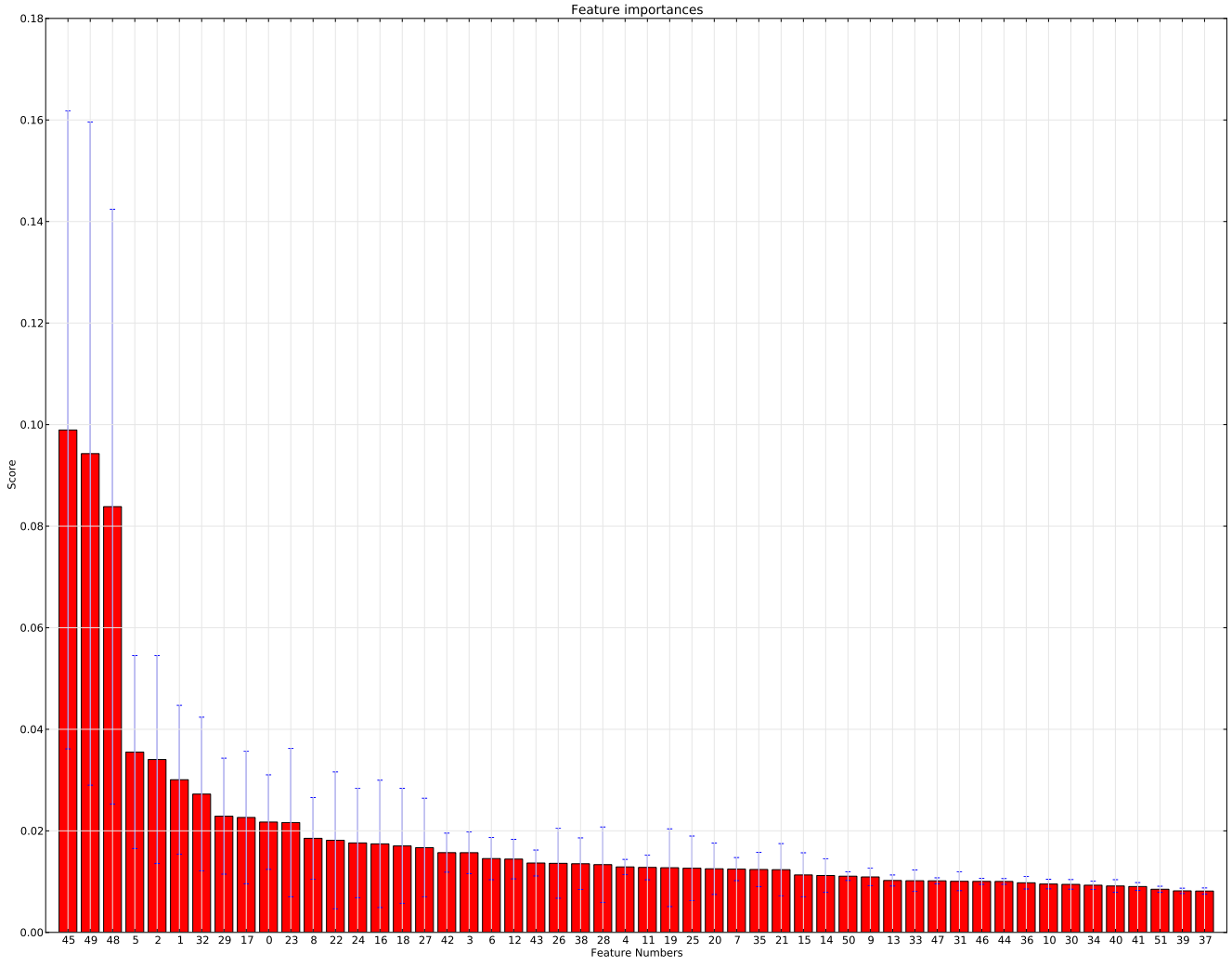



**FIGURE C.3.** Tree-based Feature Selection for main class using 300 images

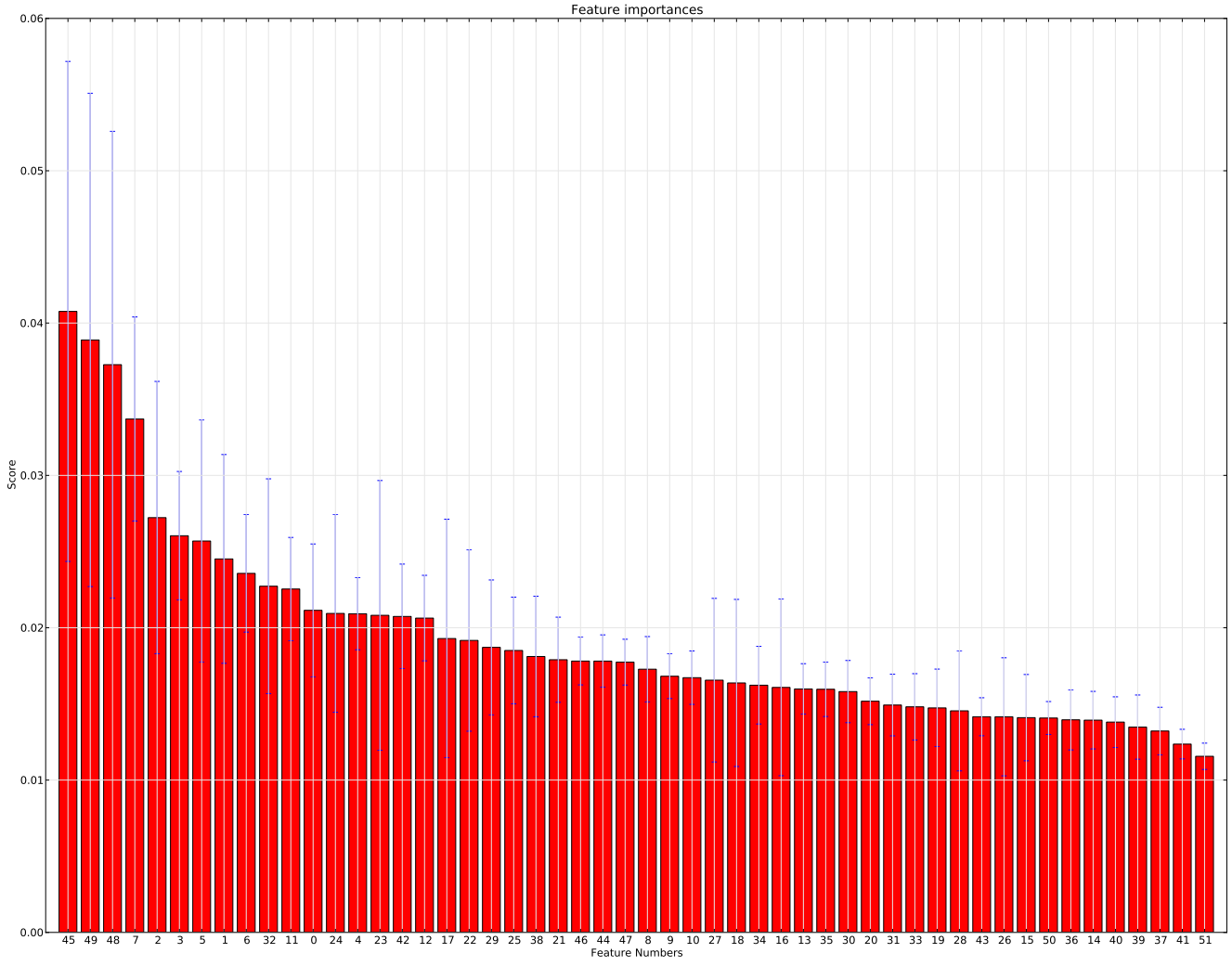



**FIGURE C.4.** Tree-based Feature Selection for sub class using 100 images

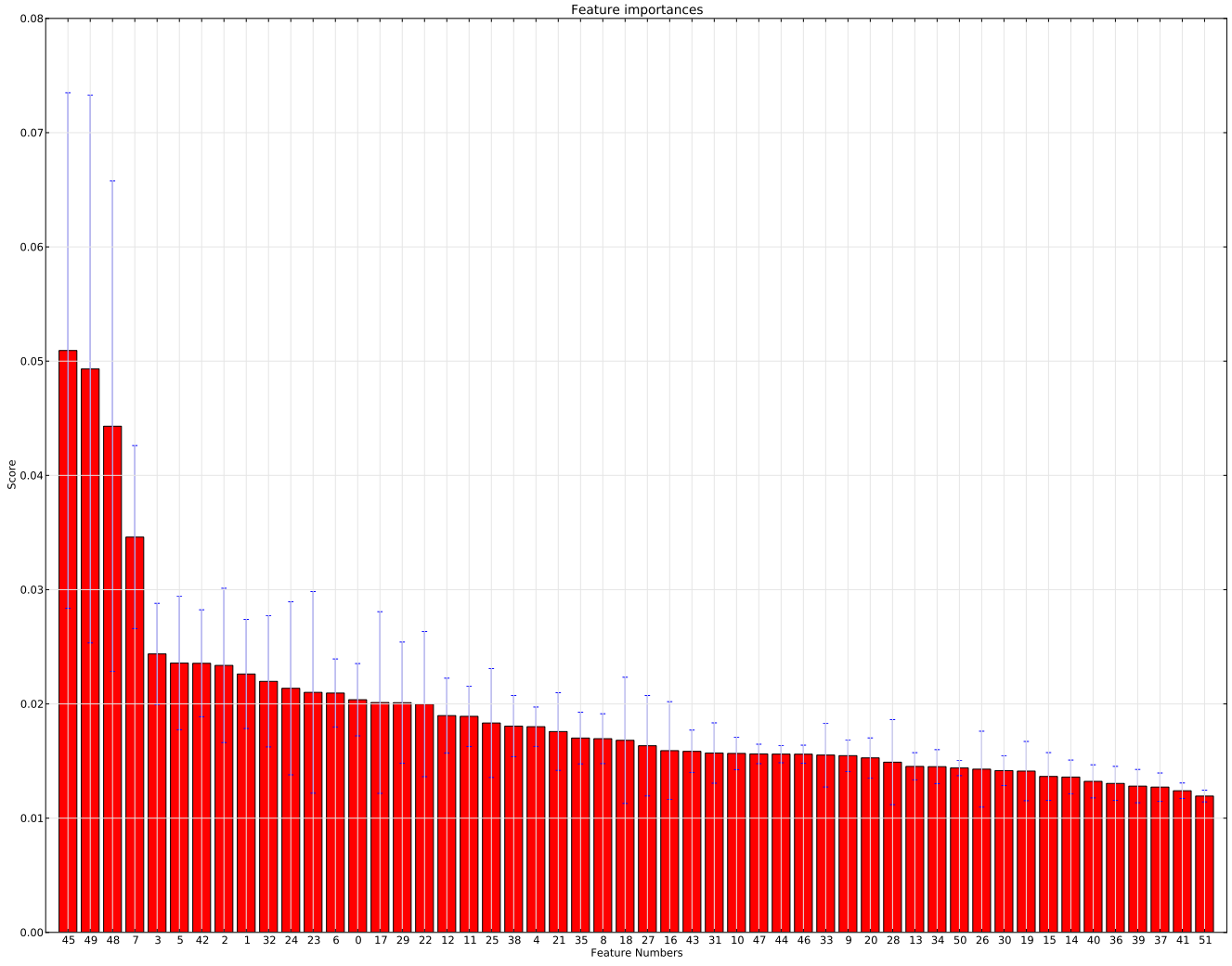



**FIGURE C.5.** Tree-based Feature Selection for sub class using 200 images

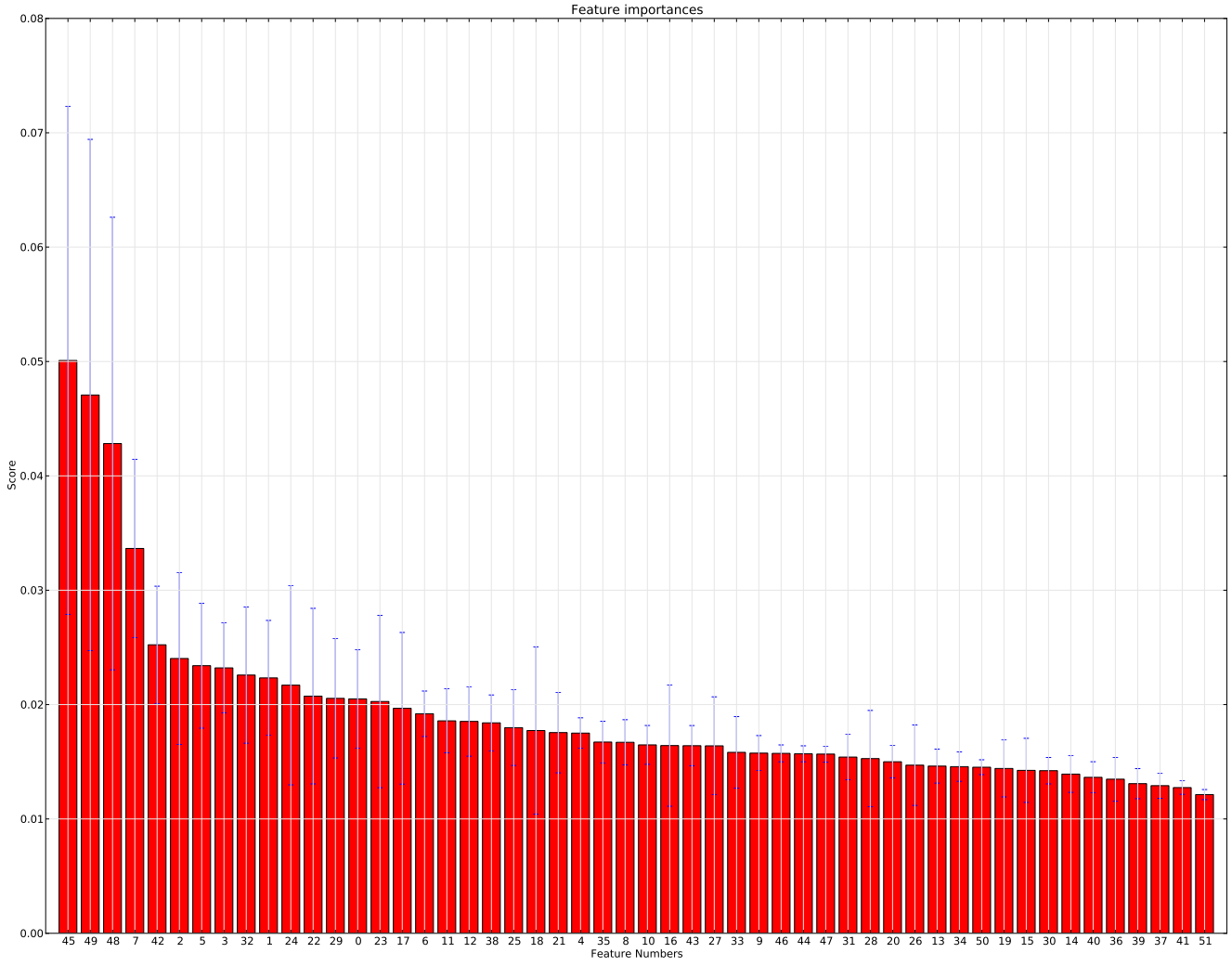



**FIGURE C.6.** Tree-based Feature Selection for sub class using 300 images