Convolutional Networks in Scene Labelling

Ashwin Paranjape
Stanford
ashwinpp@stanford.edu

Ayesha Mudassir
Stanford
aysh@stanford.edu

Abstract

This project tries to address a well known problem of multi-class image segmentation of indoor scenes. The objective is to label every pixel in a scene with the category of the object it belongs to. We seek to do this in a supervised setting without the need for manually designed features by using convolutional networks.

1. Introduction

Multi-class scene segmentation is a well studied problem in computer vision. While most approaches rely on various feature selection algorithms, we are investigating the problem of extending the convolutional neural net framework to achieve per pixel labeling of a given scene. We train and test our implementation on the Stanford background dataset[4]. We present two possible end-to-end fully convolutional networks for achieving scene segmentation. The first approach is the one followed by Farabet et.al. [2] which uses multi-scale convolutional networks trained on pixel data to extract dense feature vectors that encode information for labeling each pixel in an image. In the second approach, we try to propose an architecture which is like the deconvolutional neural net, but is trainable.

2. Related Work

Scene segmentation has been approached with variety of methods in the past. Most methods still rely on CRF, MRF or similar types of graphical models to ensure consistency in labelling. Many methods also rely on pre-segmentation of image into superpixels and extract features form individual segments to get the most consistent labels for the image. A few methods in the recent past have started using convolutional networks to train the network from end-to-end and and produce a powerful representation of features [5]

Our key inspiration was due to work done by Farabet et.al. on learning hierarchical features for scene labeling [2] where they train a multiscale convolutional network on raw pixels and aim to extract dense feature vectors that encode regions of multiple sizes centered on each pixel. With this approach they achieve 78.8 % accuracy. However this method alone does not accurately pinpoint the object boundaries. They augment their method with three different approaches from conventional vision literature, involving super-pixels, conditional random field over those superpixels and multilevel segmentation with class purity criterion. Simply averaging class predictions over superpixels and using CRFs solves the problem of pinpointing object boundaries. Finally multilevel segmentation allows for a richer tree-based segmentation and optimizes the segmentation process with a global objective of class purity.

Simonyan's paper [6] also states that backpropagated gradients on image are equivalent to using deconvolutional networks by Zeiler et.al. [7] wherein, they reverse the layers and apply them on the generated code. One remarkable aspect is that they use the maxpooling masks to unpool. In an attempt to combine the regenerative prowess of deconvnets with the ability to train on per pixel class labels, we propose a new network architecture.

3. Approach

The two approaches we followed for image segmentation are as follows:

3.1. Graph based classification on multiscale convolutional networks

This is a direct implementation of the approach in the paper by Fabaret et. al. [2]. There are two separate tasks in this approach. The first task is to train a convolutional neural network to learn and extract multiscale features in an image and the second task is to use additional supplementary methods such as use of super pixels, conditional random fields to improve the predictions. We only targeted the first task in the project. The two tasks in short are described here

3.1.1 Multiscale feature extraction

This task is concerned with extracting features for further processing with graph based methods. We require a transform $f : R^P \to R^Q$ where the image patch is a point in R^P and it is mapped to R^Q where it can be classified linearly. Convolutional neural nets have the advantage of learning features useful for this transform. However if there are pooling/subsampling layers, then they degrade the ability of the model to precisely locate objects. On the other hand, without pooling/subsampling layers, the window considered for convolution is small, offering a poor observation basis and is unable to capture long range interactions.

To counter these disadvantages, a multiscale convolutional network extends the concept of spatial weight replication to the scale space. From an input image I, a multiscale pyramid of images X_s is constructed. Each scale is treated through Conv-ReLU layers with the same weights shared across scales. Before the last fully connected layer, images at all scales are upscaled to the original size. Now a linear classifier predicts the class label for each pixel based on upscaled output volumes across different scales. In another method the linear classifier is simply replaced by a two layer network. The architecture is as shown in the Figure 1.

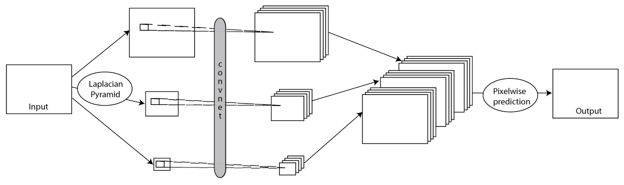

Figure 1. Multiscale Convolutional Neural Network Architecture

3.1.2 Scene Labelling Strategies

There are three strategies, each better than the previous one, which would improve the predictions as obtained from the multiscale convolutional network.

Superpixels We use the method proposed by [3] to generate an oversegmentation of the image. We simply compute the average class distribution within each superpixel.

Conditional random fields A graph $G = (V, E)$ with vertices $v \in V$ and edges $e \in E \subseteq V \times V$. Each pixel is associated to a vertex and edges are added between every two neighboring nodes. The CRF energy term is given by

$$E(l) = \sum_{i \in V} \phi(\hat{\mathbf{d}}_i, l_i) + \gamma \sum_{e_{ij} \in E} \Psi(l_i, l_j)$$

The unary term $\phi(\hat{\mathbf{d}}_i, l_i) = e^{-\alpha \hat{\mathbf{d}}_{i,a}} \mathbf{1}(l_i \neq a)$ Here $l_i = \arg\max_{a \in \text{classes}} \hat{\mathbf{d}}_{i,a}$, $a \in$ classes and $\hat{\mathbf{d}}_i$ is the normalized exponential of the class scores for each pixel over all classes. . The pairwise term consists of $\Psi(l_i, l_j) = e^{-\beta ||\nabla \mathbf{I}||_i} \mathbf{1}(l_i \neq a)$, where $||\nabla \mathbf{I}||_i$ is the l_2 norm of the gradient of the image I at pixel i

Parameter free multilevel parsing A family of segmentations, in particular a segmentation tree, is analyzed to automatically discover the best observation level for each pixel in an image. The segment associated with each node in the tree is encoded by a spatial grid of feature vectors pooled in the segment's region. This is done by means of a shape-invariant attention function, which is essentially a mask over the segment followed by pooling over a fixed size grid to produce a uniform sized descriptor. A classifier is then applied over the descriptor to produce a histogram of categories. Impurity is now defined as the entropy of the historgram. Each pixel is labeled by the minimally-impure node amongst the ancestors of the pixel in the tree.

3.2. Supervised Convnet-Deconvnet Framework

The above method does not make use of spatial domain information in its conv-nets. In that approach there are no pooling layers, however due to independent processing on smaller scales, the problem of smaller convolutional windows is partly mitigated. It is however not clear if it affords the same advantages as a deep convolutional network with pooling layers. We try to exploit the advantages of pooling in our convnet and propose the architecture as shown in Figure 2.

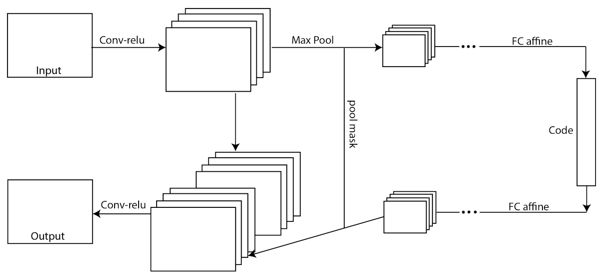

Figure 2. Convolutional-Deconvolutional Neural Network Architecture

The first part of this architecture is the convolutional layers, [conv-relu-pool]XN followed by a fully connected affine layer. At this point, we have the features representing the image. We call them the code corresponding to the image. These features would be used to get the segmentation of the image using the remaining part of the architecture. The final part of the architecture is the deconvolutional layers. Again, as before, we have a fully connected affine layer. We train this architecture such that the final output after the deconvolutional layers is the segmented image. We use the pooling mask, as used by Zeiler et.al. [7] to upscale the data in the deconvolutional network. Also, on pooling we loose more than 75% of the data. We need to bring back this information at every level in order to perform proper per-pixel labeling. For this purpose, we concatenate the hidden layers of first part of the architecture to the upscaled hidden layers at every step. Another advantage of concatenating the outputs of the Conv-ReLU layers of the first part of the architecture with the second part of the architecture is that every filter in the first part of the architecture has two sources of gradients. Just like the GoogleNet this would make the training go faster despite the apparent depth of the complete network.

To justify our architecture, if we compare the multiscale network with the proposed network, we observe that if in the first part of the network, instead of pooling the output of the previous Conv-ReLU layer, we simply used the original image. And similarly in the second part of the network, instead of feeding the output of upscale layer to the next Conv-ReLU network, we simply concatenate with the output of the next Conv-ReLU layer, we get an architecture which is very similar to a multiscale network.

4. Technical Details

We did not use Caffe or any other deep neuralnet based framework for our project as there are no predefined layers in caffe for upscaling as required in the multiscale network. Similarly, the convnet-deconvnet approach was quite unique, involving concatenation as well as upscaling, not available in Caffe. Instead, we started from the assignment code as the base. The major technical details in our implentation are listed below:

4.1. Parametric Server

We wish to categorize each pixel in an image to one of the given classes. The forward and backward passes for each of the networks were quite slow. Due to this, training these networks was a difficult task. We solved this problem by using the parametric server approach as was used by Google. We trained over a cluster of 20 CPUs. However, forking the process on multiple CPUs results in a large memory overhead from creating replicas of processes. Instead we used the ipython parallel computing library[1], which creates a cluster of python interpreters which require only the specific data on which each CPU needs to work. We achieved a speedup of about 20 by using this parametric server based approach.

4.2. Layer Dictionary

We defined a layer dictionary to easily extend the architecture to as many layers as required without having to change the code at all. The dictionary was used to specify the network architecture to be used and it implemented the forward and backward passes based on this dictionary. Each layer could be initialized using different parameters like weight scale, bias scale, number of filters, filter size, pool size, pad size, stride size, etc as and when appropriate.

4.3. Max vs Two-norm

We recognized that while we were trying to use the mask from the pool layer, in the upscale layer, to generate the larger image, the mask (which is essentially argmax) is inherently discontinuous. We propose to rectify it by replacing the maxpool layer with a softer max. Max is essentially $\infty-$norm. We plan to replace it with softer measures like $2-$norm and use these coefficients for the upscale layer.

5. Experiment

We used the Stanford Background Dataset[4] to test, train and validate our proposed networks. The Stanford Background Dataset consists of 715 images having an approximate size of 320-by-240 pixels, each pixel labelled into one of the following categories: sky, tree, road, grass, water, building, mountain, foreground, unknown. In addition we added one more category: undefined, to take into account the variable image size: if the image was of size 300x240, then the last 20 columns of the resultant image (of size 320x240) were labeled as undefined. Thus, our goal was to classify each pixel in an image into one of the 10 categories. We used the first 429 out of the 715 images for training, and equally split up the remaining (143 each) for validation and testing. Although this may appear to be a small dataset, we are training for each pixel and so, effectively, the number of train, validation and test points are 32947200, 10982400 and 10982400 respectively (since image size is 320x240). To visualize our results, each of the 10 labels was given a unique greyscale color.

At first, we tested and checked if the gradients from both the network architectures were correct. In the multiscale approach, we experimented with Gaussian and Laplacian Pyramids. The laplacian pyramid gave us better results and we sticked to them from then on. After this, most amount of time was spent in hyperparameter optimization. This was difficult because of the complex architecture. We played with different combinations weight initializations, window sizes in the convolutional layers, learning rates, regularization, learning rate decay and momentum to get the best possible validation accuracies. We started with a relatively high learning rate and learning rate decay, and a low momentum to begin with for the first 50 epochs. The next 20 epochs after these had a smaller learning rate decay and a greater momentum.

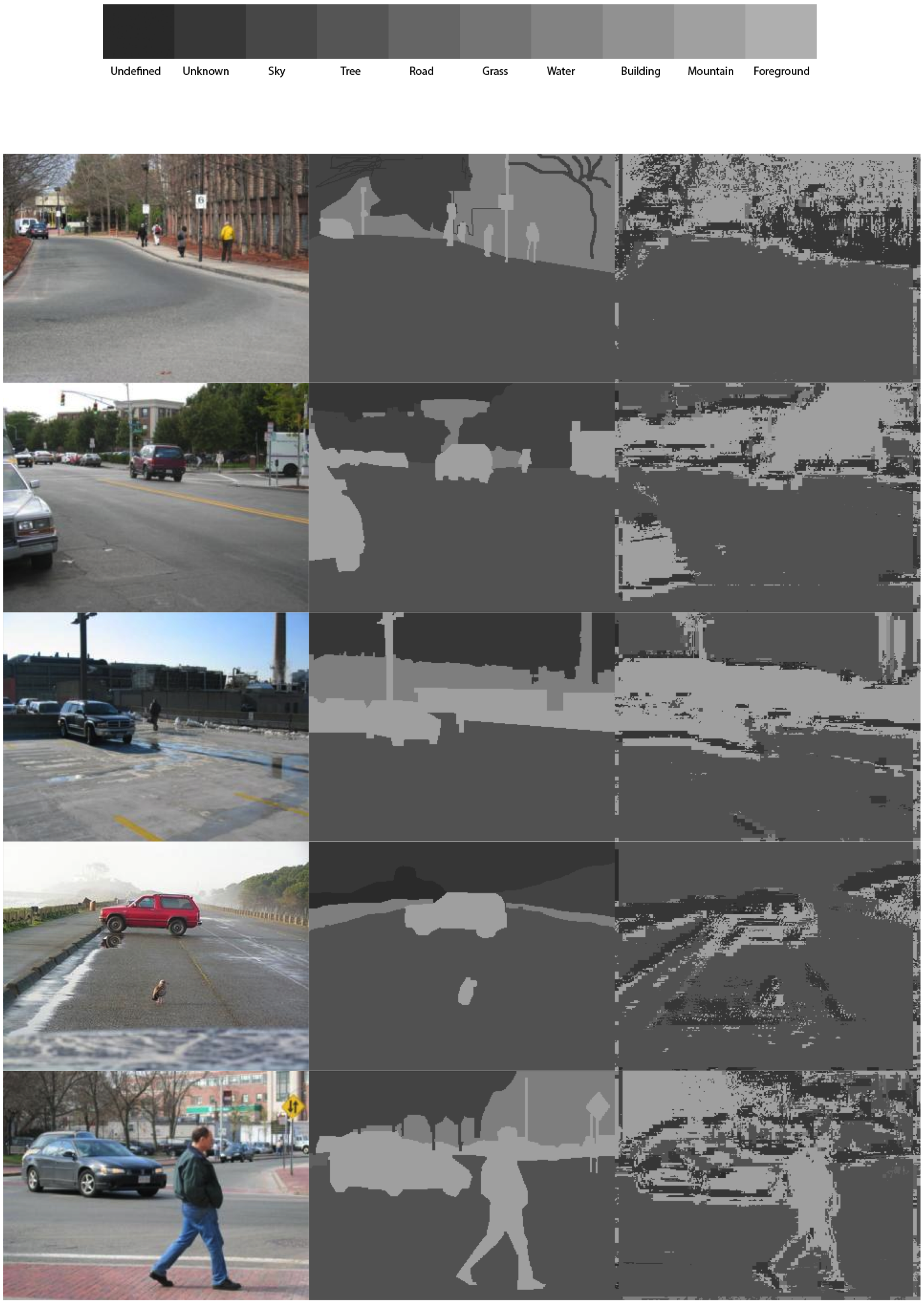

Figure 3. Top five segmentation results from multi-scale architecture. (L-R): Input image, Ground truth labels, Output learnt labels

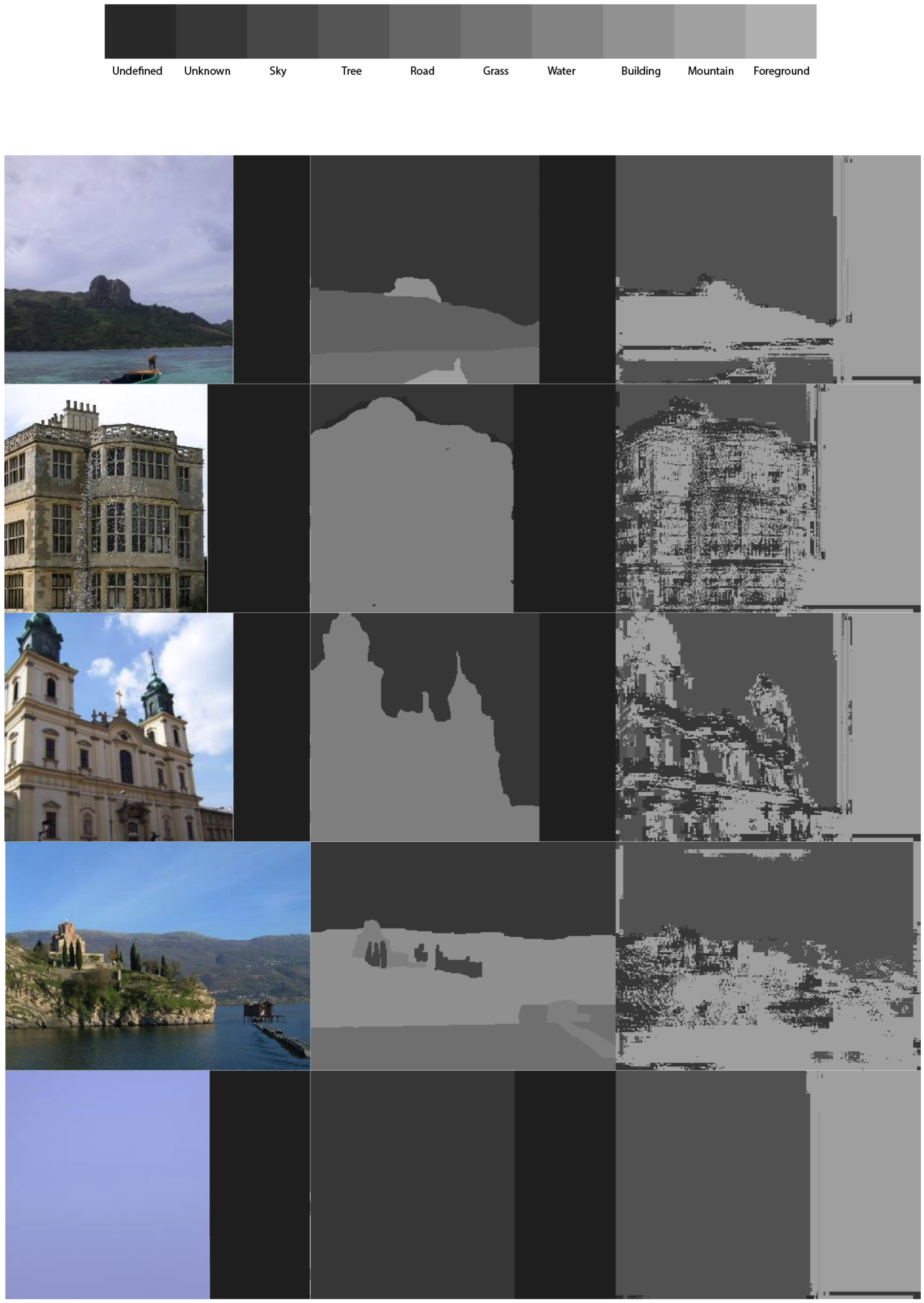

Figure 4. Bottom five segmentation results from multi-scale architecture. (L-R): Input image, Ground truth labels, Output learnt labels

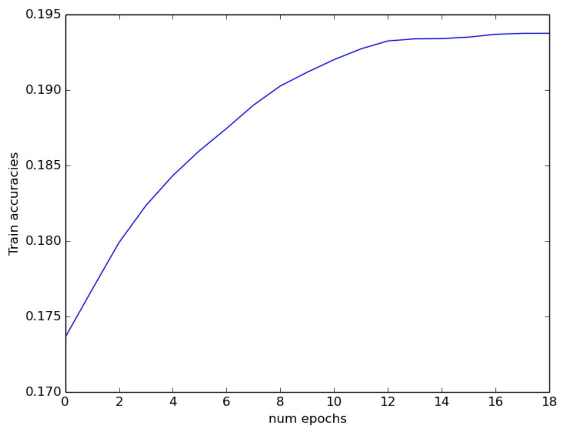

Figure 5. Train accuracy from epcoh 50 to 70

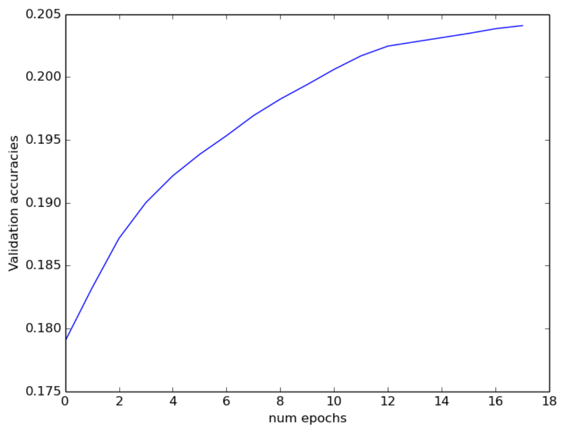

Figure 6. Validation accuracy from epoch 50 to 70

We evaluate our results based on per pixel and per class accuracy. We achieved an average per pixel accuracy of only 20% on the validation data and 23% on the test data after training for about 70 epochs. Figure3 and Figure4 show the top and bottom five images on the basis of per pixel accuracies. It was hard to know the exact reason for such low accuracies. As shown in Figure3, the segmentation is quite good and is close to the ground truth. However, looking at the worst five predictions based on average accuracies, it appears as though, even though the boundaries seem to be correct, the class labels are completely wrong. Figure7 shows the the cross correlation between the actual and the predicted labels and Table1 shows the per class accuracies.

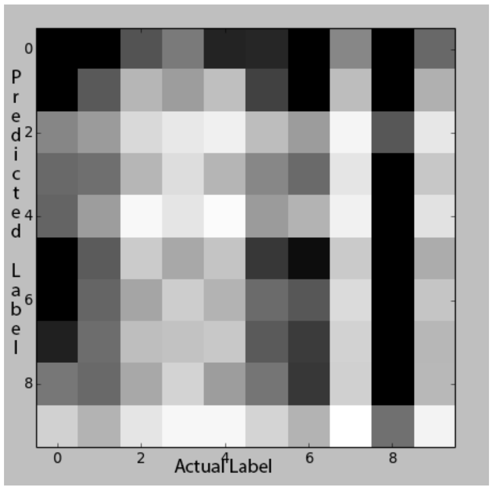

Figure 7. Cross-correlation prediction matrix between classes (in log scale)

Class	Accuracy
Undefined	0.0
Unknown	00.17
Sky	03.90
Tree	31.94
Road	43.46
Grass	00.02
Water	00.04
Building	41.92
Mountain	0.0
Foreground	11.54

Table 1. Per Class Accuracies

We see that only a few classes are predicted with accuracies more than 0.1. The rest have very less accuracies. For example, Tree, Road and Building is predicted fairly accurately. Whereas grass, water, mountain and sky have neglibible accuracies. We believe that this is because of a the training being stuck in a local minima. Initially it was hard to get more than two dominant predicted classes. However with careful training, we were able to expand it to 4 label classes. We hope that with better tuning, we should be able to improve upon these results.

Another lapse, we believe, was that we could not add a second convolution layer. The cluster simply crashed after we did that. Thus we believe we were unable to extract higher level features from the training data and hence we were unable to predict other classes.

Although we did implement conv-deconv, we did not get any good results, mainly because we were limited by the number of epochs we could execute it for. Even with a parameter server, and small batch sizes for each CPU, each

CPU required memory in excess of 10GBs and would crash after some time.

6. Conclusion

We implemented two architectures, one existing and one novel. We were able to train Multiscale Convolutional Neural Network Architecture to a fair degree. However were not successful in training Supervised Convnet-Deconvnet Framework. We believe that with use of optimized code for training, we should be able to achieve satisfactory results.

References

[1] Ipython.parallel. `http://ipython.org/ipython-doc/dev/parallel/`.

[2] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1915–1929, 2013.

[3] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[4] S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semantically consistent regions. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1–8. IEEE, 2009.

[5] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014.

[6] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[7] M. D. Zeiler, G. W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025. IEEE, 2011.