Scene parsing on SIFT Flow sub-dataset

Can Wang
Stanford University
Stanford, CA
canw@stanford.edu

Abstract

The report introduces the problem, the method I used, the running results and my analysis. To do scene parsing on the SIFT Flow sub-dataset, I implemented several models, from the simplest convolutional neural networks to the pre-segmentation based laplacian pyramid multi-scaled convolutional neural networks. I constructed these models by starting from a base model and gradually complicating it by adding pyramid or pre-segmentation. Finally I tested these models, compared their results and drew some conclusions from it.

1. Introduction

The problem I investigate is scene parsing, also known as full-scene labeling, i.e. labeling each pixel in the image with the category of the object it belongs to. It is important for understanding the meaning of an image. This problem has been approached with kinds of methods in recent years. These methods can be classified to be two categories, non-parametric methods and learning-based methods. Non-parametric method is mainly to transfer the label from existing neighbor images in training data to an image in test data based on dense scene alignment[4] or superpixels[6]. As for learning-based methods, more and more methods are using neural network or deep learning. Some methods use small convolutional neural network as feature extractor and further use superpixels, CRF model, or optimal purity cover to do pixel-wise classification[2]. Some methods use deep fully convolutional networks, upsampling and multi-scale predictions fusing to get dense prediction.[5]

2. Problem statement

I used a subset of the SIFT Flow dataset to do scene parsing. The SIFT Flow dataset consists of 2,688 images(256*256) with pixel labels for 33 semantic categories(”building”, ”grass”, ”tree”, etc.) and 3 geometric categories(”horizontal”, ”vertical”, and ”sky”). The dataset is split in 2,488 training images and 200 test images. As geometric categories are kind of trivial compared to the semantic categories, I only focused on predicting semantic categories in this project. Due to limitation of computer memory and running time, I extracted a subset from the complete SIFT Flow dataset. The subset consists of 360 images(256*256) which are split into 273 training images, 60 validation images and 27 test images. After trained, the model is supposed to output pixel semantic labels for the test images. By comparing the results with ground truth, we can see how this model works.

The evaluation metrics I use includes per-pixel accuracy and average per-class accuracy. Per-pixel accuracy is defined as the fraction of the number of pixels classified rightly over the number of pixels to be classified in total. Average per-class accuracy is defined as the average of per-pixel accuracy of all the classes existing in the dataset (This may be smaller than 33 since not all the classes will show up in a part of images). The per-pixel accuracy of one class is the fraction of the number of pixels belonging to the class that are classified rightly over the number of all the pixels belonging to that class.

Some state-of-the-art performances evaluated by per-pixel accuracy and average per-class accuracy on SIFT Flow dataset is listed in Table 1. They are not very comparable with my results on the sub-dataset considering that I use much less training data and less testing data than these papers. It can be an approximate guide but not too much. Besides, my objective is mainly comparing the results between the different models I implemented, and gaining interesting result and valuable intuition from the comparison, instead of just working for higher and higher performance.

3. Technical Approach

I tried mainly four kinds of models. The first model is a naive Fully Convolutional Neural Network with an upsampling layer and a pixel-wise softmax classifier. The second model combines RGB-YUV conversion and laplacian pyramid with the ConvNet similar as the the first one. The third one uses the former two models to extract features and

Method	per-pixel acc.	ave. per-class acc.
Liu et al.[4]	74.75	-
Tighe et al. [6]	76.9	29.4
Farebeat et al. [2] 1	72.3	50.8
Farebeat et al. [2] 2	78.5	29.6
Long et al. [5]	85.1	51.7

Table 1. Results on SIFT Flow. Liu is a non-parametric label transfer method via dense scene alignment. Tighe is a non-parametric label transfer method via superpixels. Farebeat is a multi-scale convnet trained with two sampling methods: balanced frequencies(1), natural frequencies(2).

layers	input size	output size
conv7-16	$N \times 3 \times 256 \times 256$	$N \times 16 \times 256 \times 256$
relu	$N \times 16 \times 256 \times 256$	$N \times 16 \times 256 \times 256$
pool2	$N \times 16 \times 256 \times 256$	$N \times 16 \times 128 \times 128$
conv7-32	$N \times 16 \times 128 \times 128$	$N \times 32 \times 128 \times 128$
relu	$N \times 32 \times 128 \times 128$	$N \times 32 \times 128 \times 128$
pool2	$N \times 32 \times 128 \times 128$	$N \times 32 \times 64 \times 64$
upsampling	$N \times 32 \times 64 \times 64$	$N \times 32 \times 256 \times 256$
conv1-33	$N \times 32 \times 256 \times 256$	$N \times 33 \times 256 \times 256$

Table 2. Architecture of FCNN for model 1

feed the features to a pre-segmentation-based small convolutional neural network. The fourth model essentially has the same architecture with the third one. However it is not a combination of pre-trained model with a small ConvNet but a unified model that can be trained together.

3.1. Model 1

Model 1 is a fully convolutional neural network(FCNN). The data belonging to the training dataset, validation dataset and test dataset are all preprocessed(zero-centered) by subtracting the mean of training images before fed into the FCNN. Comparing to ordinary CNNs, FCNN converses all the affine layers to be convolutional layers, hence FCNN can operate on an input of any size and produce an output of corresponding size. For pixel-wise dense classification, we need the size of output to have exactly the same spatial size with the input (only different in channel size). The architecture of my FCNN is in Table 2.

I realized the upsampling layer as a backwards strided convolution process. For example, if I want to upsample an image by 4 times, it is in fact the backward process of convolution with filter size being 4×4 and stride being 0. By realizing the upsampling layer with this method we can also learn the upsampling weight matrix during the training process. Suppose the input is a four-dimensional array of size $N \times C \times H \times W$, where N is the image number dimension, C is the channel or feature dimension, and H and W are spatial dimensions. The output is an array of size $N \times C' \times H' \times W'$. We need to upsample the input by d times, where d is an integer. So $H' = d \times H$ and $W' = d \times W$. The upsampling matrix M will have size of $C \times C' \times 4 \times 4$. For the n^{th} input data at pixel (h, w), we can compute

$$\sum_{i=1}^{C} Input[n, i, h, w] \times M[i, :, :, :],$$

which is an array with size of $1 \times C' \times 4 \times 4$. In total we can get $N \times H \times W$ such kind of arrays. By concatenating those arrays according to the same spatial relationship as the input, we can get the output.

Since the last layer of the FCNN consists of 33(corresponding to the number of semantic classes) 1×1 filters(padding 0, stride 1), after the conv1-33 layer, for each input image, we get 33 maps and each map has the same spatial size with the input image. The 33 values at each pixel location are the scores corresponding to the 33 categories of that pixel. Then we use the multiclass cross entropy to form the data loss, which leads to a pixel-wise softmax classifier. Suppose the scores of categories a for a pixel i is $s_{i,a}$. Then the normalized predicted probability distributio for the pixel is

$$\hat{p}_{i,a} = \frac{e^{s_{i,a}}}{\sum_{b \in categories} e^{s_{i,b}}}.$$

The loss for each pixel is measured by

$$-\sum_{a \in categories} p_{i,a} ln(\hat{p}_{i,a}),$$

where $p_{i,a}$ is the true target probability distribution of pixel i: $p_{i,a} = 1$ if pixel i is labeled a, and 0 otherwise. Then the total data loss is

$$L_{data} = -\sum_{i \in pixels} \sum_{a \in categories} p_{i,a} ln(\hat{p}_{i,a}).$$

After adding l2-norm regularization, we get the total loss. By minimizing the loss through mini-batch gradient descent, we can get the weight of the network. When predicting, we set the category of pixel i to be

$$argmax_{a \in categories} s_{i,a}$$

.

3.2. Model 2

Model 1 predicts the category of a pixel only based on the neighbor pixels. But sometimes faraway pixels can also influence the prediction result. Multiscale convnet can solve this problem. First I did a color space conversion from RGB to YUV. Since it is approximately a linear transformation

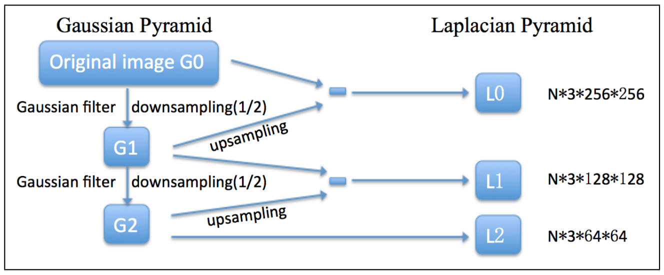

Figure 1. Structure of Laplacian Pyramid

layers	output size
conv7-16	$N \times 16 \times 256 \times 256$
relu	$N \times 16 \times 256 \times 256$
pool2	$N \times 16 \times 128 \times 128$
conv7-16	$N \times 16 \times 128 \times 128$
relu	$N \times 16 \times 128 \times 128$
pool2	$N \times 16 \times 64 \times 64$
upsampling by 4 times	$N \times 16 \times 256 \times 256$
conv1-33	$N \times 33 \times 256 \times 256$

Table 3. Architecture of FCNN for model 2.1

I think this doesn't change the total model space, but this may make it easier or harder to find a better solution. After the conversion, I transform the images into three scales (in octaves) through a Laplacian pyramid, whose structure is shown in Figure 1. Then the Y, U and V channels of each scale in the pyramid are independently locally normalized, such that each local 8×8 patch has zero-mean and unit variance.

3.2.1 Model 2.1

For this model we only use the first scale of the laplacian pyramid, i.e. L0 in Figure 1, to be input for a FCNN similar to that in Model 1. L0 has the size of $N \times 3 \times 256 \times 256$. The architecture is shown in Table 2.

3.2.2 Model 2.2

Each scale of the laplacian pyramid is fed to one FCNN almost the same as that in Model 2.1. L0 is fed to a FCNN exactly the same as that in Model 2.1 while the FCNNs that L1 and L2 are fed into are only different in the upsampling layer. L1 and L2 are fed into FCNNs with "upsampling by 8 times" and "upsampling by 16 times" respectively. Weights for the same layers are shared between the three FCNNs. Since different input sizes correspond to different upsampling sizes, the three FCNNs all output score arrays of the same size, which is $N \times 33 \times 256 \times 256$. Then they are added together to be the final score and used to compute the multiclass cross entropy data loss.

layers	output size
conv5-32	$N \times 32 \times 256 \times 256$
relu	$N \times 32 \times 256 \times 256$
conv1-33	$N \times 33 \times 256 \times 256$
pre-segmentation	$N \times 33 \times 256 \times 256$

Table 4. Architecture of FCNN for model 2.1

3.3. Model 3

Predicting the category of pixel independently may get result which lacks spatial consistency. We will use pre-segmentation (one kind of superpixel) to solve this problem. First we get segmentations for all the images in our dataset with the method in [1]. With the method, we can set a parameter to determine how fine the output segmentation is.

We can get features from the pre-trained models above. For Model 1 and Model 2.1, the output of upsampling layers(with size of $N \times 32 \times 256 \times 256$ and $N \times 16 \times 256 \times 256$ respectively) are features we need. For Model 2.2, we get the features(with size of $N \times 48 \times 256 \times 256$) by concatenating the outputs(with size of $N \times 16 \times 256 \times 256$) of upsampling layers of the three FCNNs together. The features all has the same image number dimension and spatial dimension as the input. Then we fee the features into a small ConvNet, whose architecture is in Table 4.

According to the pre-segmention, we have got some idea about whether some pixels belong to one group or not. In the pre-segmentation layer, we compute the average score of the pixels belonging to one group and assign the average score to be the score of every pixel belonging to the group. Then we can compute the cross entropy data loss based on the averaged scores.

By using different pre-trained models to get features, we construct different model. Model 3.1, Model 3.2, Model 3.3 uses features from pre-trained Model 1, Model 2.1, Model 2.2 respectively.

3.4. Model 4

Model 4 uses the same architecture as Model 3 but it trains the whole model simultaneously instead of train a part of model to extract features and then train another part of model to get scores from the features. Considering Model 3.1 has higher performance than Model 3.2 and Model 3.3. I only implemented Model 4.1, corresponding to Model 3.1, whose architecture is shown in Table 5. I trained one Model 4.1 with random initialization and another one with the weight parameter from Model 3.1(a.k.a. warm start).

4. Results

The per-pixel accuracies and average per-class accuracies on test data of these models mentioned above are listed

layers	input size	output size
conv7-16	$N \times 3 \times 256 \times 256$	$N \times 16 \times 256 \times 256$
relu	$N \times 16 \times 256 \times 256$	$N \times 16 \times 256 \times 256$
pool2	$N \times 16 \times 256 \times 256$	$N \times 16 \times 128 \times 128$
conv7-32	$N \times 16 \times 128 \times 128$	$N \times 32 \times 128 \times 128$
relu	$N \times 32 \times 128 \times 128$	$N \times 32 \times 128 \times 128$
pool2	$N \times 32 \times 128 \times 128$	$N \times 32 \times 64 \times 64$
upsampling	$N \times 32 \times 64 \times 64$	$N \times 32 \times 256 \times 256$
conv5-32	$N \times 32 \times 256 \times 256$	$N \times 32 \times 256 \times 256$
relu	$N \times 32 \times 256 \times 256$	$N \times 32 \times 256 \times 256$
conv1-33	$N \times 32 \times 256 \times 256$	$N \times 33 \times 256 \times 256$
pre-seg	$N \times 33 \times 256 \times 256$	$N \times 33 \times 256 \times 256$

Table 5. Architecture of FCNN for model 4.1

Method	per-pixel acc.	ave. per-class acc.
Model 1	0.4122	0.0803
Model 2.1	0.4511	0.0843
Model 2.2	0.4255	0.0792
Model 3.1	0.5783	0.1108
Model 3.2	0.5272	0.1036
Model 3.3	0.4168	0.0769
Model 4.1	0.4459	0.0866
Model 4.1(warm start)	0.5853	0.1117

Table 6. Performance on test data.

in Table 6.

From the results we can see that for every model, the per-pixel accuracy is much higher than average per-class accuracy. This is because the data-loss focuses on per-pixel accuracy. The model is trying to predict the pixels which belong to "BIG" class rightly and doesn't care about the accuracy of "SMALL" class.

5. Analysis

5.1. upsampling layer

In Model 4.1, the upsampling matrix has size of $32 \times 32 \times 4 \times 4$. The visualization of the 32 4×4 matrix corresponding to the first channel of input and 32 channels of output respectively is shown in Figure 2. The visualization of the 32 4×4 matrix corresponding to the 32 channels of input and first channel of output respectively is shown in Figure 3. As the upsampling layer is in working on high-dimensional feature space. The visualization is kind of abstract. But at least we can see that the layer does give degree of freedom to the whole model. It actually make the process of upsampling learnable.

5.2. multi-scale convnet

My expectation is that the model using multi-scale laplacian pyramid should outperform the simple FCNN model.

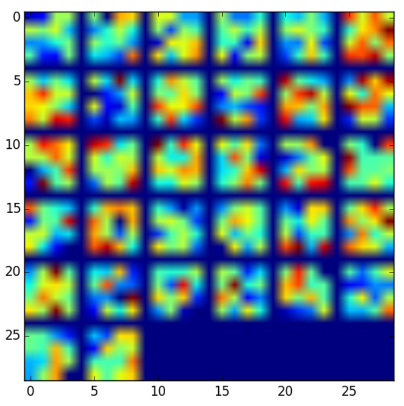

Figure 2. upsampling weight corresponding to the first channel of input and 32 channels of output

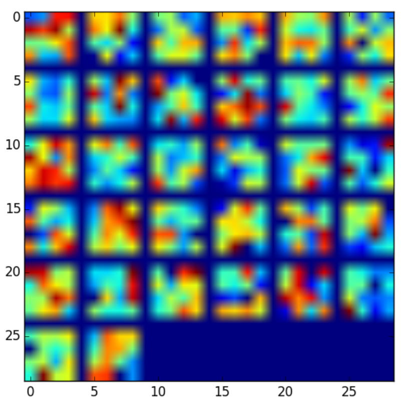

Figure 3. upsampling weight corresponding to the 32 channels of input and first channel of output

From the results of Model 1 and Model 2.1, it is true. However, from the result of Model 3.1 and Model 3.2, it is false. Model 1 uses the information of the raw pixel itself while Model 2.1 uses L0 as input and mainly utilized the high-frequency information around a pixel to determine its category. The result pre-segmentation improves the performance of Model 1 more than that of Model 2.1 may be very data-dependent. Let's look at a picture Model 3.1 classified poorly in Figure 4 $\sim$10. Figure 4 is the picture and Figure 5 is the pixel-wise classification ground truth. Different locations of small sub-pictures represents different categories. Figure 6 is the pre-segmentation for the picture. Comparing Figure 5 and 6 we can find that the pre-segmentation fits quite well with the ground truth, which means this pre-information is really able to help. Figure 7 $\sim$ 10 are the

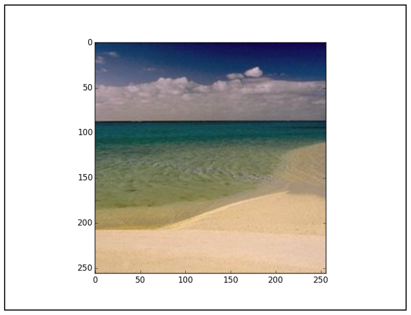
Figure 4. example picture

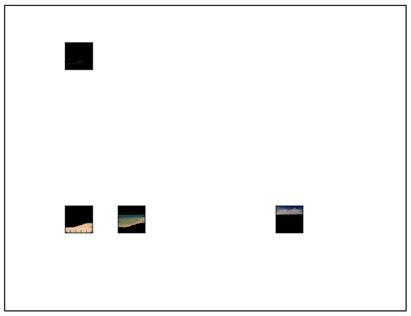
Figure 5. ground truth

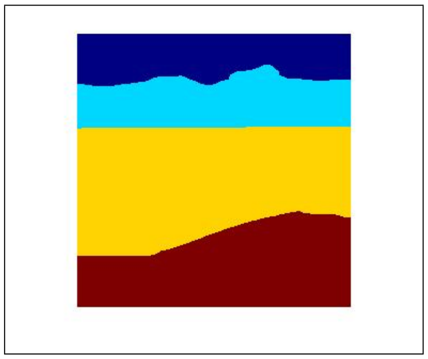
Figure 6. pre-segmentation of example picture

classification result of Model 1, Model 2.1, Model 3.1 and Model 3.2 respectively, and the corresponding per-pixel accuracy is in the name of figures. We can see that the results of Model 1 and Model 2.1 are essentially similar to each other but the pixel classes are more diffuse in the result of Model 1. However, the specific number of pixels belonging to different groups will affect the result of Model 3.1 and Model 3.2. Hence Figure 9 and 10 have huge difference.

We can see that for this specific example Model 3.2 has better performance than Model 3.1, while the total test accuracy of Model 3.2 is lower than that of Model 3.1, which means the result is actually data-dependent, or maybe more precisely feature-dependent. Within these test images, more images' classification results, essentially the feature maps,

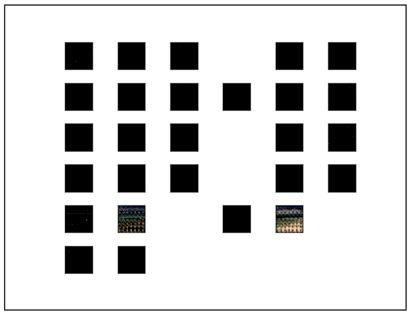
Figure 7. classification result of Model 1, per-pixel accuracy 0.3513

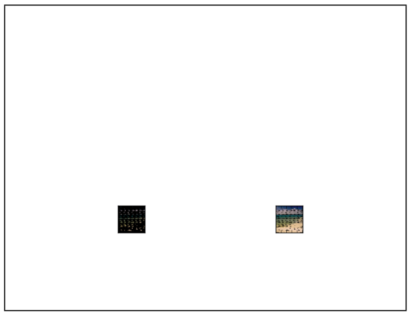
Figure 8. classification result of Model 2.1, per-pixel accuracy 0.3772

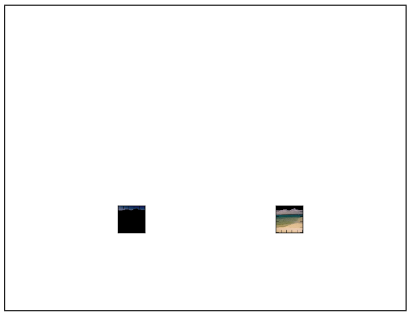
Figure 9. classification result of Model 3.1, per-pixel accuracy 0.1787

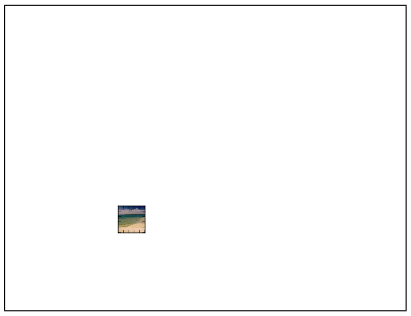
Figure 10. classification result of Model 3.2, per-pixel accuracy 0.3837

of Model 1 collaborate better with pre-segmentations than that of Model 2.1, not like the example above.

Besides, we can see that Model 2.1 outperforms Model 2.2 here. However, in fact, Model 2.2 includes Model 2.1. On the one hand, this means the Model 2.2 is harder to be trained well. This may be caused by overfitting. Because there are more parameters it can adjust to gain low loss and high training accuracy, the model is easier to stuck in some point where training accuracy looks good but in fact relying too much on specific data. We could try further tuning the training parameter of Model 2.2 to see if the results can be better or try data augmentation to see it will work. For pixel-wise dense classification, data augmentation is kind of different with whole-picture classification. But we can still use random crops(on both input image and ground truth) or color jittering. On the other hand, this also means that for the dataset we use, high frequency information(L0) seems to be more useful than low frequency information(L2). It's kind of intuitive because edge and contour are important for segmentation.

5.3. pre-segmentation and self-consistency

Adding pre-segmentation is able to improve self-consistency and probably able to improve both per-pixel accuracy and average per-class accuracy. We can see this very clear from the example above.(Figure 4 $\sim$ 10) But when I run the program, I find that it is easier to stuck around some not-very-good point in the process of training. By "stuck" I mean that the training accuracy and validation accuracy stay the same. A good solution to this problem is: not to run too many iterations one time, but run many times. Since each time we run the program, we use random initialization, which can lead us to different stuck point. Among these points, we can find a good one.

Besides, we notice that Model 3.3 has slightly worse performance than Model 2.2, which means pre-segmentation even make things worse.I think since multi-scale model tries to capture the influence of a pixel?s far-neighbor-pixels on its category, it should indeed be harder to cooperate with pre-segmentation.

5.4. pre-training vs. big model

We can see that Model 3.1 outperforms Model4.1 although Model4.1 includes model3.1. On the one hand, Model 4.1 is deeper and harder to find proper hyperparameters. On the other hand, model4.1 may rely too much on pre-segmentation and hence its optimal result for training data is harder to generalize for testing data. Besides, due to limit of memory, the batch-size is only 1 or 2, which may be worse for Model 4.1. Because of the generalization hardship, when the weight changed after some SGD steps, the training accuracy even drops a lot. But we can see that warm start did help a lot. Although warm start doesn't give us much better performance, at least it proves that the big model doesn't overfit and it can approach some good point. Besides, it gives us a way to train big model in the future. We can first split big model to small models and train them separately. Then we can use the pre-trained results to be a warm start for the big model.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[2] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, August 2013.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[4] C. Liu, J. Yuen, and A. Torralba. Nonparametric scene parsing: Label transfer via dense scene alignment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(12), December 2011.

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

[6] J. Tighe and S. Lazebnik. Superparsing: Scalable nonparametric image parsing with superpixels. *ECCV*, pages 352–365, 2010.