A Brief Introduction on STL-10 Image Recognition Task

Yining Gao

April 2016

1 Introduction

STL-10 `https://cs.stanford.edu/ acoates/stl10/` is an image dataset containing 10 classes. In addition to its 4,000 training examples, it contains 100,000 unlabeled images that contain our ten classes and also images outside of those classes. The primary challenge of this task is to leverage this unlabeled data to improve supervised model.

2 Pre-processing and Data Augmentation

2.1 Pre-processing

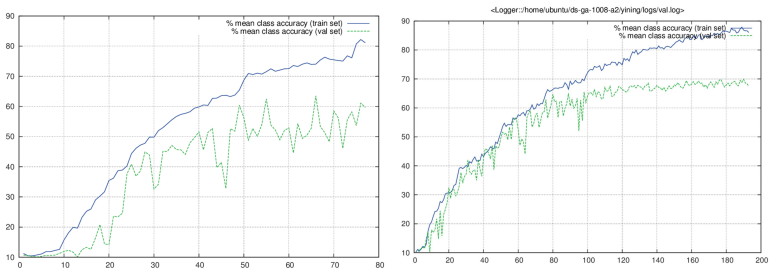
- First, images are mapped into YUV space to separate luminance information from color information.
- The luminance channel (Y) is then locally normalized using contrastive normalization operator. Other color channels are normalized globally.

2.2 Data Augmentation

I augmented the data to artificially increase the size of the dataset using the following methods:

- **Transformation:** vertically and horizontally translate by a distance within 0.1 of the patch size
- **Flip:** flips image horizontally or vertically
- **Rotation:** rotate image by an angle up to 20 degrees
- **Rescale:** rescale original image by a factor within [0.7,1.4][2]. If the rescaled image is smaller than the original, resize it. If larger, crop it to match the original size
- **Color Contrast:** multiply the projection of each image pixel onto the principal components of the set of all pixels by a factor between 0.5 and 2

Data augmentation slightly boosts the performance of the model. The charts below show the increase in performance on two models:



Left: original vgg **Right**: vgg with data augmentation

3 Unsupervised Methods

3.0.1 K-means

Compared to Auto-Encoder, K-means clustering is very fast and thus easily implemented at large scale. The idea is simple:

- In pre-training, K-means algorithm finds cluster centroids that minimize the distance between data points and the nearest centroid.
- In supervised-training, centroids are then used as the initialization of first layer filters in the ConvNet.

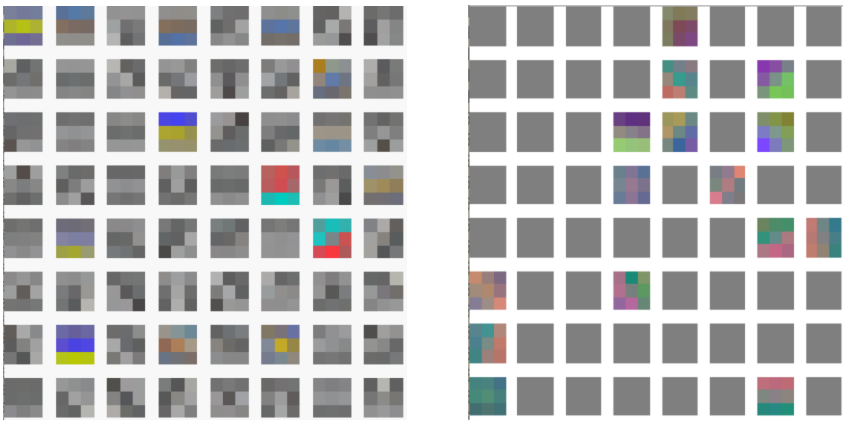
There are several issues with K-means algorithm. [4] mentioned that simple K-means introduces redundancy in filters. To avoid this problem, I set the number of centroids twice the number need, and choose half base on image gradient. Also, in [3], it mentioned that centroids tend to be biased by the correlated data and thus whitening is essential.

After several experiments, I finally go with the patch selection and pre-processing approach described below:

- Randomly select 50000 3*3 size patches from unlabeled extra data base on image gradient. I set the image gradient threshold to be mean+3.5*std.
- Pre-processing: Use a guassian kernal to locally normalize each channel. Use ZCA whitening normalized patches. Choose $\epsilon = 1e - 2$. Note that setting this number too small can cause high-frequency noise to be amplified and make learning more difficult.
- Run K-means on the these patches and get the centroid.
- Randomly select half of the centroids as weight initialization. Again, I set a image gradient threshold to be average.

A good weight initialization increase model accuracy significantly. Using the above stated initialization method, sample model got 20% accuracy on training and validation in the first epoch, and jumped to about 70% on training data before 15 epoch.

Here's the visualization of centroids from K-means algorithm , in comparison to the first layer filter from ConvNet, which is initialized with this set of centroids. K-means learned low-frequency , edge-like features from the unlabeled data:



Left: centroids(3*3). **Right**: 1st layer filter visualization

A natural idea comes into mind is that whether our model accuracy will continue to increase if I feed the first layer ConvNet(with filter weights initialized from K-means) with unlabeled data, get the output feature map, run K-means again on the output, and use the obtained centroids as the initialization of the second layer ConvNet. I implemented this thought, the result, however, is not satisfactory.

I also tried using centroids of size 5 by 5, the latter gave a slightly worse performance than 3 by 3.

3.1 Pseudo-label

The idea of Pseudo-label is quite intuitive: picking up the class which has the maximum predicted probability, are used as if they were true labels [5].

The author got good results running this method with very small labeled data on the MNIST handwritten digit dataset. In our assignment, however, classification problem is more difficult and we even has a proportion of unlabeled data that don't belong to 10 classes.

I modified this method in the following way:

- Feed all unlabeled data into trained ConvNet (sample model),select 1000 sample with highest score, add output class prediction as label and add them to training data. By this step, our model accuracy improved from 73% (sample model) to 76%.
- I tried feed all unlabeled data to run the new model again, select 2000 sample with the highest score, repeat the same process, the result is not satisfactory.

3.2 Overall Architecture and Results

Finally I decided to go with K-means initialization method without using pseudo-label. For data augmentation, I use a combination of real time augmentation and pre-training augmentation. I tried three different architecture:

- 7 layers sample code architecture : 3 $\rightarrow$ 64, maxpooling(4,4,4,4), 64 $\rightarrow$ 128, maxpooling(3,3,3,3),128 $\rightarrow$ 256, maxpooling(2,2,2,2), 256 $\rightarrow$ 256, maxpooling(2,2,2,2), followed by 3 256 $\rightarrow$ 256 and a maxpooling layer. This architecture gives validation accuracy 77.9%.
- Another 7 layers architecture:3 $\rightarrow$ 64, maxpooling(4,4,4,4), 64 $\rightarrow$ 128, 128 $\rightarrow$ 128 maxpooling(3,3,3,3),128 $\rightarrow$ 256, 256 $\rightarrow$ 256, maxpooling(2,2,2,2), 256 $\rightarrow$ 512, 512 $\rightarrow$ 512 , maxpooling(2,2,2,2). This architecture gives validation accuracy 78.5%.

- 8 layers architecture: 3 → 64, maxpooling(4,4,4,4), 64 → 128, 128 → 128 maxpooling(3,3,3,3),128 → 256, 256 → 256, maxpooling(2,2,2,2), followed by 3 256 → 256 and a maxpooling layer. This architecture gives validation accuracy 78.4%

3.3 t-SNE visualization

The images below represent our attempts to use t-sne to visualize the clustering of our data. The first two images come from the fully-connected layer, and the image on the right is taken using one of the feature maps in the final convolutional layer.

References

[1] Universum Prescription: Regularization Using Unlabeled Data
`http://arxiv.org/pdf/1511.03719v5.pdf`

[2] Discriminative unsupervised feature learning with convolutional neural networks
`http://arxiv.org/pdf/1406.6909.pdf`

[3] Learning Feature Representations with K-means `Adam Coates and Andrew Y. Ng`

[4] Convolutional Clustering for Unsupervised Learning
`http://arxiv.org/abs/1511.06241`

[5] Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks
`http://deeplearning.net/wp-content/uploads/2013/03/pseudo`$label_final.pdf$