Xijie Shuai
7074-1997-16
xshuai@usc.edu
EE569 Competition 2
May 3, 2020

Competition 2: CIFAR-10 Classification using SSL

I. Intuition

In the previous assignment I achieved 63.3% accuracy, and the parameters setting are listed in Chart 1 to 3. In this competition, my primary goal is to improve the accuracy since compared to those state-of-the-art model honestly the accuracy of SSL is too low. Also, I tried to reduce the number of parameters and decrease the running time. All the intuition, specific modification and results of different experiments will be described explicitly in the next chapter.

Parameters Setting of Module 1										
	Number of AC kernels	Need Bias	Use DC Kernel	Use Channel-wise Transform	Neighborhood Construction Kernel	Neighborhood Construction Stride	Max Pooling Window	Training Samples	Threshold1	Threshold2
PixelHop++ Unit1		FALSE		FALSE			(2, 2)			
PixelHop++ Unit2	-1	TRUE	TRUE	TRUE	(1, 5, 5, 1)	(1, 1, 1, 1)	(2, 2)	10,000	0.001	0.0001
PixelHop++ Unit3		TRUE		TRUE			None			

Chart 1: Parameter setting of module 1

Parameters Setting of Module 2									
	Number of Classes	Number of Bins	Ns	Training Samples	Encode	Clusters for each class	Alpha	Learner	
				50,000					
				12,500					
Feature Selection	-1	10	1,000	6,250	DISTANCE	3	10	LLSR	LAG
				3,125					
				1,562					

Chart 2: Parameter setting of module 2

Parameters Setting of Module 2			
n_estimators	criterion	min_sample_split	min_sample_leaf
500	Gini	20	10

Chart 3: Parameter setting of module 3

II. Experiments

Note 1: Due to time limitation, I tuned the parameters in the following experiments based on using all data in module 2 and 3. Thus, the performance under different ratio of labeled training data could be not pretty good.

Note 2: In all of the following experiments, I stored the results of each module (e.g. output of PixelHop++ units) in local disk, and read it at the beginning of the next module. The purpose of this step is to enable running each module separately and to avoid retraining the whole model caused by bugs. The cost is longer running time.

Experiment 1: Reproduce the result in PixelHop++ paper

Firstly, I want to use the parameter setting in the original work to achieve the highest accuracy, which should be around 66.81%. For the parameters not described in the paper, I used bin method in feature selection (Ns=6000), assigned 3 clusters for each class in LAG, applied random forest as decision module. I will try to improve this experiment later so the number of parameters and running time will not be presented here. I only tested the ratio=1 in module 2 and 3. **The training and testing accuracy are 82.71% and 66.52% respectively.**

Experiment 2: 3x3 or 5x5?

In previous experiment I applied 5x5 neighborhood construction size instructed by the paper. In this experiment I want to test the 3x3 neighborhood construction, which means there will be four layers in the model. Negative infinite padding will be applied here so that max-pooling will not select the padding values. The reasons for turning to 3x3 are:

(1) **Reduce the parameters** in module 1. The number of parameters is determined by number of layers, neighborhood size, and number of channels. The number of channels is controlled by neighborhood size actually, and benefit from one more pooling layer in architecture, 3x3 setting theoretically will reduce the number of parameters.
(2) **Get more diverse intermediate results** so that I can test more idea on them.

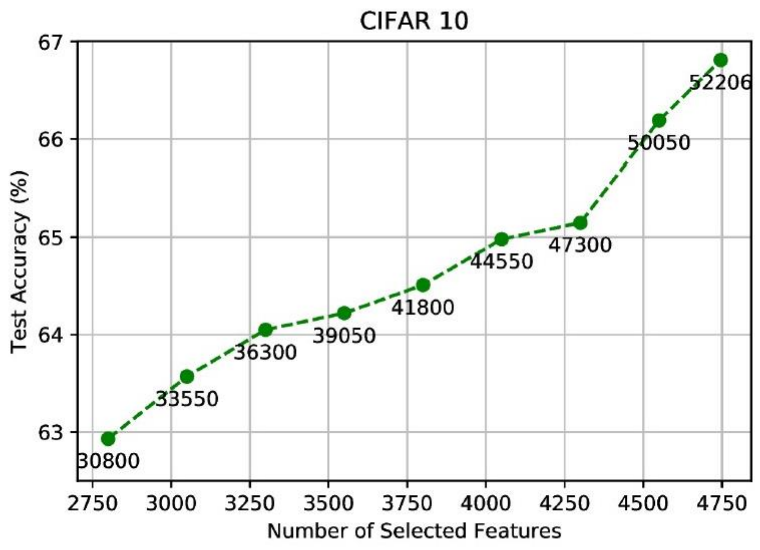
A pretty interesting finding here is, with Ns=6000, the selected channels from the output of first PixelHop++ unit is actually more than the second one (selected 2197 from 4275 channels in the output of first unit, but only 248 from 4998 channels in the second unit), which implies the features extracted from the first unit are more powerful than the second one. The training accuracy and testing accuracy are 82.8% and 66.64% respectively, **basically are same with 5x5**. This setting will be used in all following experiments. Note that different with the description in HW6 that Ns is selected features in each layers, here Ns represent the total of selected features among all layers. The reason to do that is, as described before, selecting same number of channels in some layers (e.g. layer 2) is unnecessary, and I can do this to reduce the number of parameters.

Experiment 3: Number of clusters in LAG

In this experiment I tried to reduce the number of clusters assigned for each class. Since the LLSR uses this number in calculation as a dimension (e.g. 3 clusters x 10 classes), if reducing this number will not put impact on accuracy then I can reduce the number of parameters and running time. After change it from 3 to 2, the running time of LAG decreased from 215s to 169s, and the training accuracy and testing accuracy are 82.745% and 66.7% respectively, **which is basically same which 3 clusters**. Thus, this setting will be used in following experiments.

Experiment 4: Number of selected features

In this experiment I will reduce the number of selected features for LAG. As described in the paper, reducing this number will lead to a rapid decrease as shown below. Note that reducing this parameters will not shorten the running time of feature selection module (e.g. bin method, which cost the largest part of running time in the whole model) since Ns will be used after calculating the discriminability of each channels in that module in my architecture. However, it will reduce the number of parameters in LAG. Since I don't want the accuracy dropping too much, I reduce Ns from 6000 to 4000 here, and the testing accuracy goes down to 65.74%, **which is even better compared to the original paper**. The running time of LAG reduced from 169s to 133s.



Experiment 5: Is feature selection necessary?

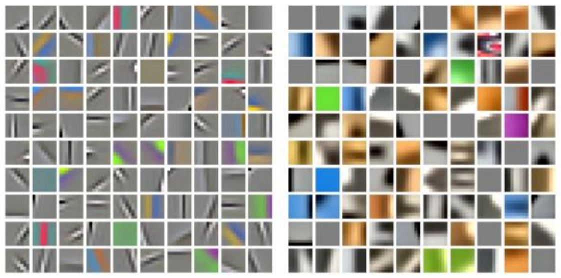
The previous experiments shows that the feature selection module takes the most of the running time (e.g. module 1 takes 395s, feature selection takes 1182s, LAG takes 133s, and module 3 takes 23s). **If I can discard the feature selection the running time will be reduced a lot**. Thus, I tried to remove the feature selections and feed the output of PixelHop++ units directly into LAG. Based on that, LAG now took 360s, which is apparently much more than before but still acceptable, and the training accuracy and testing accuracy are 85.786% and 66.63% respectively, which basically remain the same compared to experiment 3. As a summary, here I list the running time, accuracy, number of parameters under different setting below:

	Train Accuracy	Test Accuracy	Number of params	Running time
Ns=6000	82.75%	**66.70%**	~129k	~1769s
Ns=4000	80.60%	65.74%	**~89k**	~1733s
No feature selection	**85.79%**	**66.63%**	~266k	**~777s**

Note that the number of parameters are few more than the paper under basically parameters setting, but it could be better after optimization. The running time include unavoidable batch data reading and storing, as mentioned in *Note 2*.

Experiment 6: Whitening

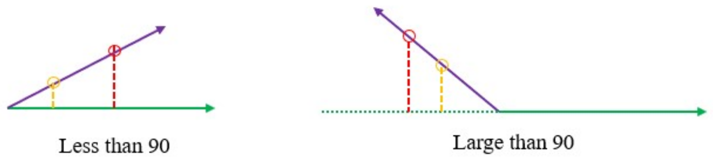
Previous proves I can reproduce the results of the original work under different parameters setting. Start from this experiment I will try to apply more tricks not mentioned in the paper. Motivated by the idea proposed in paper *an analysis of single-layer networks in unsupervised feature learning* that **whitening is important** for clustering features extracted from images by K-means since it will “yield sharply localized filters”, and they improved the accuracy by roughly 8% with this trick.


(a) K-means (with and without whitening)

Thus, I applied it in Saab transform. Since the PCA in Saab already requires zero-centering, here I only need to divide the results by their standard deviation. However, it turned out that it helped nothing in my model. The accuracy did not increase.

Experiment 7: More thought about pooling

In the neural network, since the output of layers represent the coefficient between patches and filters, and it is reasonable to preserve the strongest response on the directions of filters by max pooling. However, I thought that is not same for cascading Saab, since the direction of kernels are not guaranteed to be same with some possible features good for classification. **If the angle between PCA kernels and those features are less than 90 degrees, then max-pooling can preserve the highest response on the direction of features; but it is larger than 90, then min-pooling will actually preserve the highest response as shown below.**

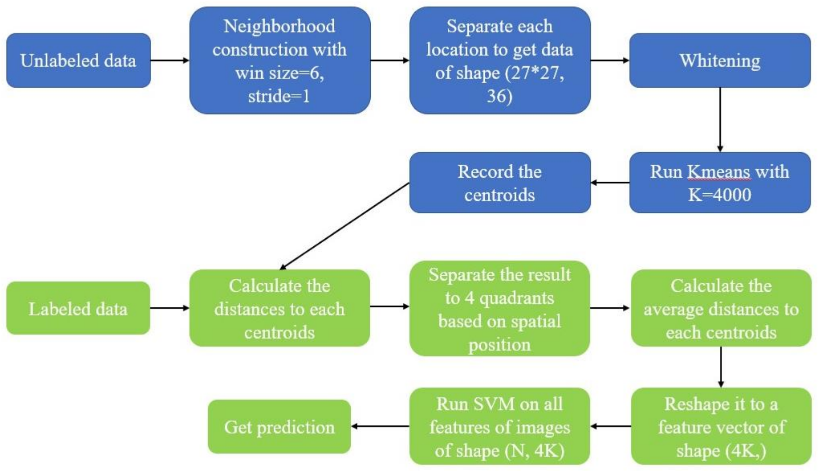


The green arrows represent the directions of possible good features for classification, and the purple arrows represent the directions of the PCA kernels. In the left one, max-pooling on PCA kernel direction will also get the strongest response on feature directions, but in the left one max-pooling takes the red sample and get the weakest response on the feature directions. The projection onto feature direction is not important, since next layer of PCA will do zero-centering and yield negative values anyway.

To apply this idea, at the pooling step I did both max and min pooling, and separated the result into train and validation set, and run random forest to examine which in max and min pooling is better for each channel for classification in the future. Apparently this step will cost a lot of time. By doing this I got 66.86% test accuracy, **which is slightly better than previous but took much more time**. I guess the reason could be the window size is so small (i.e. 3x3) and also due to the coefficient in spatial domain the values (i.e. red and orange circles, etc.) are very close, so the combination of max and min pooling will not yield big difference.

Experiment 8: Implement the paper: an analysis of single-layer networks in unsupervised feature learning

This paper was mentioned in experiment 6. The reason I want to implement it is that it also use a feed forward style and get 79.6% accuracy on CIFAR10, and published 9 years ago. It seems like that is a very good model. The pipeline is presented below to ease your understanding.


Unlabeled data
Neighborhood construction with win size=6, stride=1
Separate each location to get data of shape (27*27, 36)
Whitening
Run Kmeans with K=4000
Record the centroids
Labeled data
Calculate the distances to each centroids
Separate the result to 4 quadrants based on spatial position
Calculate the average distances to each centroids
Reshape it to a feature vector of shape (4K,)
Run SVM on all features of images of shape (N, 4K)
Get prediction

Due to memory limitation I used K=1600 for K-means and add a PCA module before decision module (i.e. SVM in the diagram above). Since SVM took incredible long time and still failed to converge in my experiment, I changed it to a random forest module. The training time is around 850s but the test accuracy is much lower than the paper, which is 51.21%. A summary of this method is presented below.

Train Accuracy	Test Accuracy	Number of params	Running time
100.00%	51.21%	**very few**	~850s

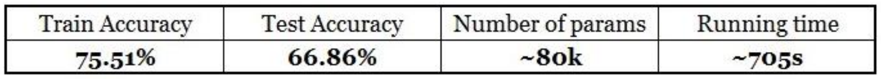
(1) Apparently there is overfitting because the gap between training and test accuracy is so huge. However, the test accuracy dropped all the time if I tried to relief that. A possible explanation for such a low test accuracy is my poor experience on random forest tuning, or, maybe SVM does really works much better here. I'm not sure about that.

(2) There are only very few parameters stored in the model: the centroids in the K-means, two numbers of whitening (i.e. mean and std), parameters in SVM or RF. PCA is optional. Thus, **the parameters of this model are much less than any model I have ever seen.**

(3) Due to memory limitation, I always separate the data into small batches since the size after neighborhood is pretty large, and even much larger after transformed into distances (size = 50000x27x27x4000 = **145 billion**, even if I use 1600 rather than 4000 the size will still be **58 billion**), and read / store / delete them one by one. Thus the running time is longer than expectation.

Experiment 9: Combine Experiment and SSL

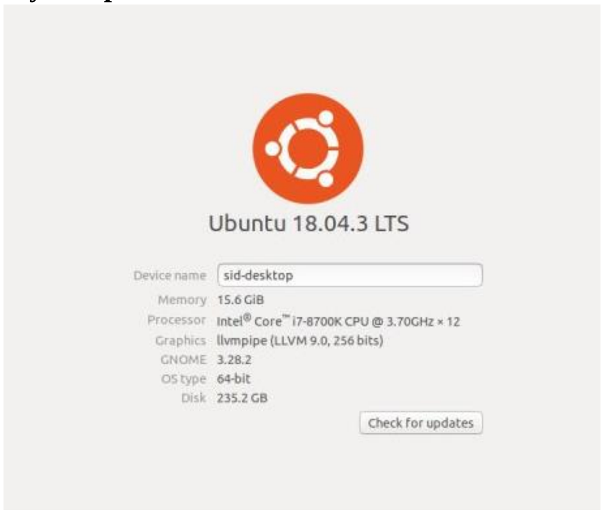
The idea in experiment 8 is basically statistic on low-level features and there is no cascading layers like in SSL. Thus, I tried to combine them together. Specifically, I applied the method in experiment 8 after the output of layer 1 and 2 (which produce low level features), so that it serves as another category of "label assisted feature reduction" modules that output the prediction of that method. Then in module 3 I combined the two sets of prediction from new method and two set of prediction from LAG from layer 3 and 4, and put all of them into a random forest. At last, I got 66.64% test accuracy, which means there is no improvement on accuracy compared to previous experiment. However, there are still some benefits: it reduce the number of parameters since LAG will not handle the output of first two layers anyone, and thanks to smaller window size (3x3 rather than 6x6) I can use a much smaller K in Kmeans 8 (specifically, 100 for first layer and 150 for second layer, there must be better parameters setting but I don't have time to try), which means much less training time than in experiment (specifically, Kmeans fit takes 59s and predict takes 17s). Also, as shown in experiment 2 that almost all features in layer 3 and 4 are selected by feature selection module, thus I can directly discard the feature selection part, which can save a lot of time. An brief summary is listed below.



Train Accuracy	Test Accuracy	Number of params	Running time
75.51%	**66.86%**	**~80k**	**~705s**

Though the model seems more complex than PixelHop++, it has fewer number of parameters because of the new method, and better running time because of no feature selection and low time cost of the new module.

III. Information about my computer

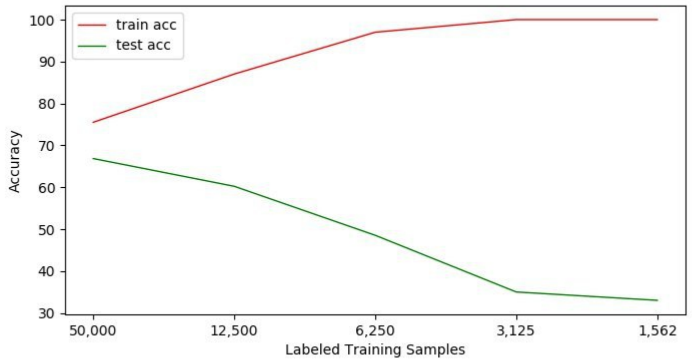


Ubuntu 18.04.3 LTS

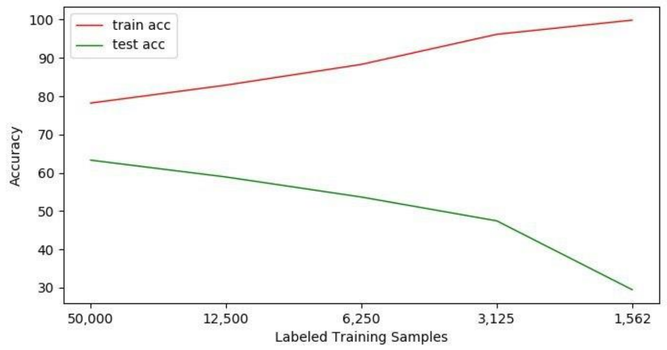
Device name	sid-desktop
Memory	15.6 GiB
Processor	Intel® Core™ i7-8700K CPU @ 3.70GHz × 12
Graphics	llvmpipe (LLVM 9.0, 256 bits)
GNOME	3.28.2
OS type	64-bit
Disk	235.2 GB

IV. Result under weak supervision

Due to time limitation I only run the Experiment 9 under different ratio of labeled training data. The result is shown below.



As mentioned in *Note 1*, I did not tune parameters for weak supervised setting. Thus, the gap between train and test accuracy gets larger and larger as fewer labeled samples are used. The reason for accuracy of Labeled Training Samples = 3125 and 1562 does not drop that fast is I can use the prediction of the new module (more specific, the first one) as the final result if the result of final random forest is too poor. Compared to the accuracy in HW6 which is shown below, the accuracy of Experiment 9 drops even faster. But again, this is because of a more complex model and poor parameter tuning. But anyway, Experiment 9 achieves a higher accuracy with much less running time and less parameters.



Another thing need to mention is that I also run some experiments which show that if I assign more clusters to classes in LAG unit I will get a better result (e.g. 67%+). However, due to the increase of the parameters I choose not to do so.