EE569 Digital Image Processing

HOMEWORK #6 Competition

Issued: 04/08/2020
Problem3 due: 05/03/2020

Yao Fu
6786354176
yaof@usc.edu

EE 569 Homework #6
May 03, 2020

Problem 3: EE569 Competition --- CIFAR10 Classification using SSL (50%)

The parameters from the Homework 6 might not be a best choice. In this competition, I will make some trials to modify PixelHop++ trained from Problem2 of Homework 6 with the purpose of improving classification accuracy, decreasing running time and reducing the model size if possible.

3.1 Abstract and Motivation

Deep learning, which is based on artificial neural networks, is one of the most popular machine learning methods. In the field of deep learning, scholars have attached vast significance to convolutional neural networks (CNNs) in recent years largely due to their outstanding performance in a great many applications. If there exists a CNN architecture, the selection of its parameters is usually generalized as a nonconvex optimization problem, which can be solved by backpropagation (BP) with efficiency. However, just as an old saying goes that “every coin has two sides”, the disadvantage of nonconvex optimization is that this procedure is totally mathematically intractable, thus stimulating individuals to design a new methodology to resolve the interpretability problem.

The professor Kuo with his students has proposed an interpretable feedforward (FF) design adopting a data-centric approach without any BP, which derives network parameters belonging to the current layer in terms of data statistics from the output of the previous layer.

After understanding the basic mechanism of feedforward-designed convolutional neural networks and successive subspace learning from the papers [1], [2], and [3] in the Homework 6, in this assignment I will make some modifications to my code with the purpose of increasing the classification accuracy.

3.2 Motivation and logics behind your design

In the traditional architecture of convolutional neural networks, the convolutional layers can implement a successive series of spatial-spectral filtering operations, during which the spatial resolutions are gradually coarser and coarser. With the purpose of tackling the trouble of the loss of spatial resolution, the authors in the paper[1] project pixels in a batch onto a set of pre-selected spatial patterns, which are obtained by the principal component analysis, and then convert spatial representations to spectral representations, thus making it possible to enhance discriminability of some dimensions. This new transform is called the Saab (Subspace approximation with adjusted bias) transform, where a bias vector is added to annihilate

nonlinearity of the activation function. In other words, the Saab transform is a new version of principal component analysis, which can be treated as a specific method to selecting anchor vector a_k and bias term b_k. The Fig. 2.1 is a simple flow diagram that can display the process of the Saab transform, which can replace the convolutional layers of the LeNet-5.

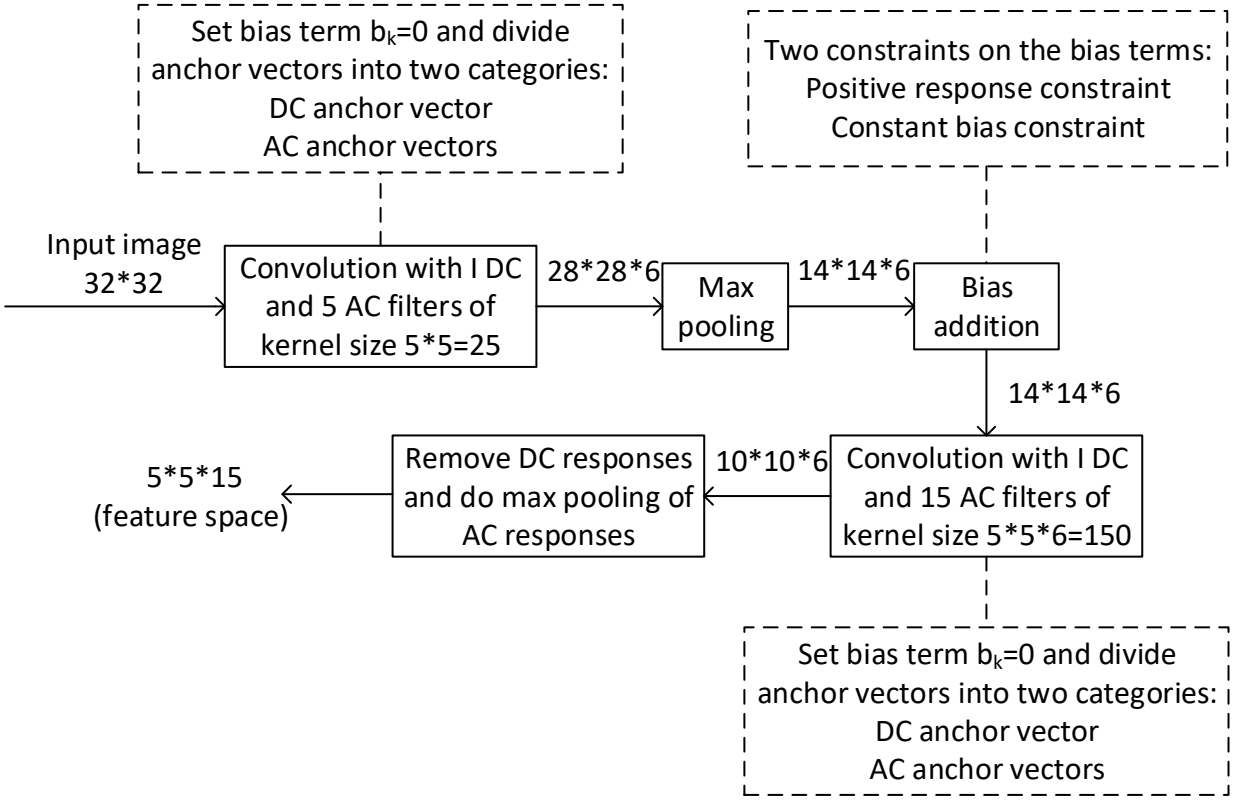

Fig. 2.1: The flow diagram of the Saab transform,
where DC anchor vector $a_0 = \frac{1}{\sqrt{n}}(1, \dots, 1)^T$,
and AC anchor vector $a_k, k = 1, \dots, K-1$.

The Fig. 2.2 is a simple framework of Successive Subspace Learning, where the functions of three modules are briefly stated. As for their processes of neighborhood construction and subspace approximation steps, I will apply the method from[3]: using the channel-wise (c/w) Saab transform in the PixelHop++ method, applying a tree-decomposed feature representation, and selecting a feature subset by the leaf node's features ordered in terms of their cross-entropy values. Specifically, the comparison between traditional Saab transform and the c/w Saab transform is shown in the Fig. 2.3. The traditional Saab transform usually takes an input of dimension $S_i \times S_i \times K_i$ and generates an output of dimension $S_{i+1} \times S_{i+1} \times K_{i+1}$ after the max-pooling operation that converts $S_i \times S_i$ into $S_{i+1} \times S_{i+1}$. The c/w Saab transform inputs K_i channel images of dimension $S_i \times S_i$ and outputs K_{i+1} images of dimension $S_{i+1} \times S_{i+1}$ after max-pooling.

In my design, I will use Python3 and codes from [4] to build a sample PixelHop++ model.

Firstly, I am about to train the module1 by using 10k training images from the CIFAR10 and save the PixelHop model. Thanks to the experience with the previous Homework 6, during

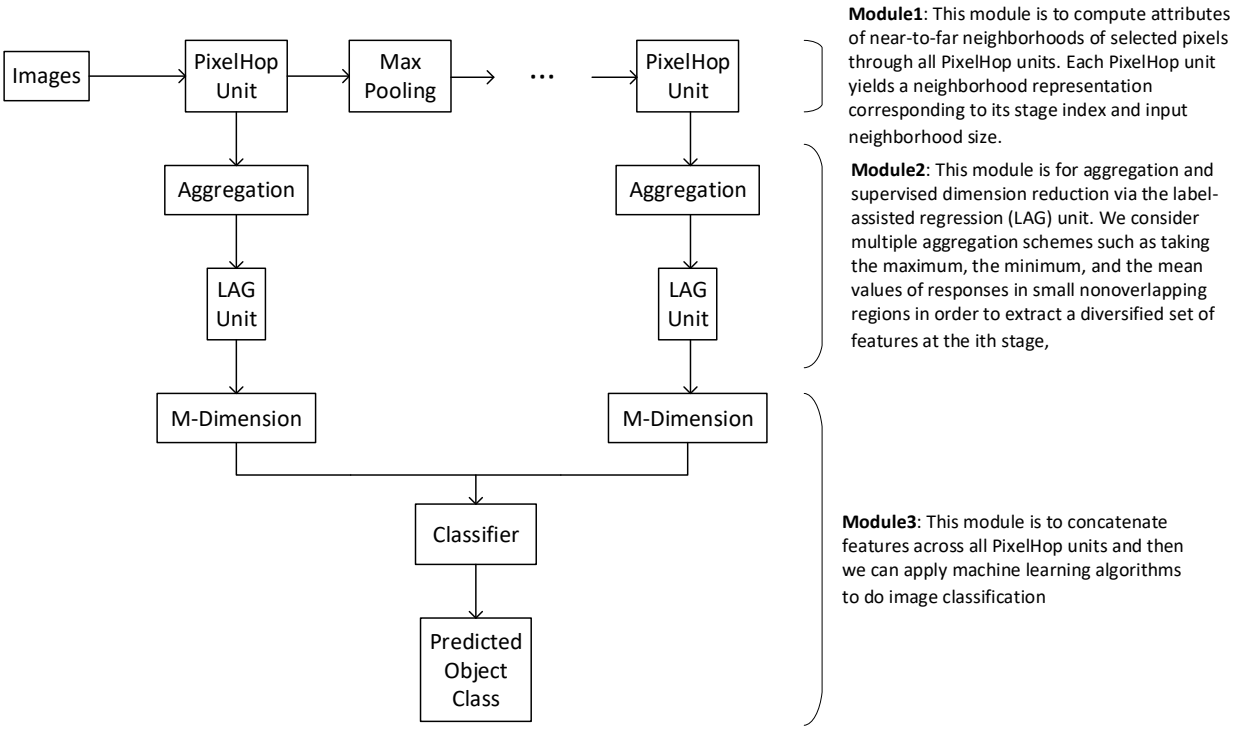

Fig. 2.2: The framework of SSL and the functions of three modules

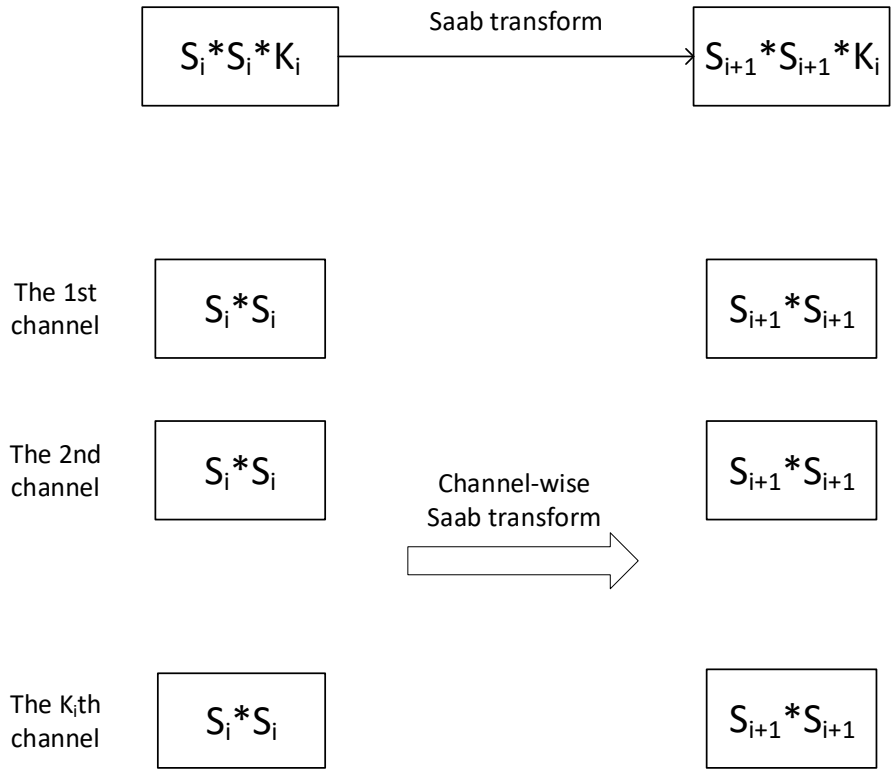

Fig. 2.3: The comparison between the traditional Saab transform and the c/w Saab transform

which I utilized the AWS to use 50k images to fit a PixelHop model, this change's effect on the final classification accuracy is trivial so that I will still use 10k images to train the model in the Module1 in this assignment. In the Module1 I will adjust the neighborhood size and energy threshold to see if this can increase test accuracy.

What's more, with the saved PixelHop model, I will train Module2&3 on different numbers of the training images. I will try increasing the number of selected features from each hop and adjust the pooling layers (max-pooling, mean-pooling, and min-pooling) to check if those changes are conductive to increasing classification accuracy.

As for saving time, according to the TA's hints, the "feature selection" part applying the K-means will tremendously increase the running time as the number of samples goes up because we have to apply the K-means algorithm to every sample! To tackle the running-time trouble, I can simply utilize the bin method instead of K-means, which is good for saving running time compared with the Problem2 of Homework6.

When mentioning the model size, I have to firstly summarize the procedure of getting my model size in order to control the model size with respect to 3 separate steps:
Step1: Find out the number of Saab filter coefficients in the PixelHop++ units.
Step2: Find the number of coefficients of the regression matrix in the LAG units. With the suggestion of the TA, the parameter of the regression matrix can be viewed as

$$M \times (n+1),$$

where n is the feature dimension and the M is the output dimension of LAG.
Step3: According to the TA's advice, I can approximate my classifier as a linear regression classifier. The numbers of hops, classes, and output dimensions of LAG should be considered.

Later I will discuss my experiment results compared with the Problem2 of HW6 respectively based on classification accuracy, running time, and model sizes and make a summary.

3.3 Classification accuracy with full and weak supervision

Table 3.1: The test accuracy of the PixelHop++ model of different numbers of training samples in Module2&3 with the K- means in feature selection

Samples	All	1/4	1/8	1/16	1/32
Test accuracy	0.6512	0.5915	0.5205	0.4028	0.1302

Table 3.2: The test accuracy of the PixelHop++ model of different numbers of training samples in Module2&3 with the Bin method in feature selection in the case 1

Samples	All	1/4	1/8	1/16	1/32
Test accuracy	0.6311	0.5545	0.4483	0.3175	0.1383

Table 3.3: The test accuracy of the PixelHop++ model of different numbers of training samples in Module2&3 with the Bin method in feature selection in the case 2

Samples	All	1/2	1/4	1/8	1/16	1/32
Test accuracy	0.6575	0.6122	0.5163	0.3748	0.1402	0.1759

In the case of the Table 3.1 and the Table 3.2, I use the K-means to do feature selection as compared with the bin method. As shown in the Table 3.1, the final test accuracy is 0.6512 that is higher than 0.6311 in the Table 3.2. However, the bin method will highly reduce the running

time that will be discussed later. The error analysis for the situation in the Table 3.1 and the error analysis for the situation in the Table 3.2 are shown in the Fig. 3.1 and the Fig. 3.2 respectively.

For the case in the Table 3.2, when I do min-pooling, the test accuracy is 0.6415 and when it comes to mean-pooling, it is 0.6299. The min-pooling can, to some extent, increase the accuracy, which I believe there is some occasionality, but the mean pooling will not only decrease the test accuracy but will also slow the speed down largely, which should be totally abandoned. From my limiting experiments, changing the threshold value and size of spatial neighborhood fails to be beneficial for increasing test accuracy. For example, when I set the neighborhood size as 7 or change the TH1 as 0.0015, the final test accuracy just goes down a bit compared to the original case. However, when I select more features in every unit (changing 1750 to 3000 in the first unit, changing 1000 to 2000 in the second unit, and changing 250 to 275 in the third unit), my test accuracy could go up in a certain amount as shown in the Table 3.3 compared with the Table 3.2. The test curve for the Table 3.3 is shown in the Fig. 3.3, indicating that the test accuracy will decrease fast and there might be some fluctuations when the number of training samples becomes smaller. Compared with the result of HW5 Problem2 whose result is shown in the Table 3.4, the BP-CNN's test power is stronger and more robust because I apply a very deep neural network and the model in this assignment is still in its infancy.

Table 3.4: The experiment results with different number of the train samples in BP-CNN

Number of Train Samples	10k	30k	50k
Test Accuracy	0.5417	0.7578	0.7924

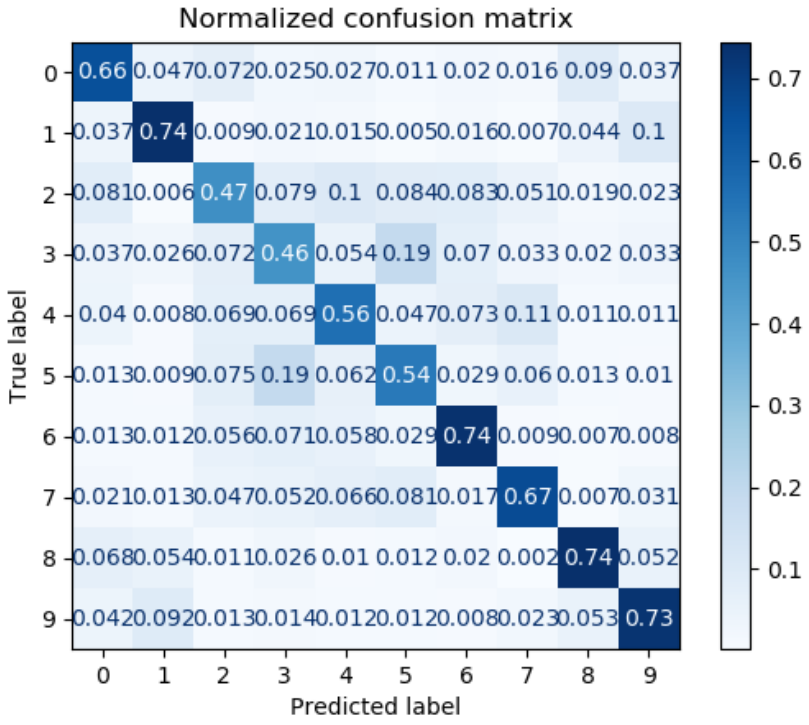

Fig. 3.1: The normalized confusion matrix
when the number of train images in Module2&3 is 50k by the K-means,
where airplane 0, automobile 1, bird 2, cat 3, deer 4, dog 5, frog 6, horse 7, ship 8, and truck 9

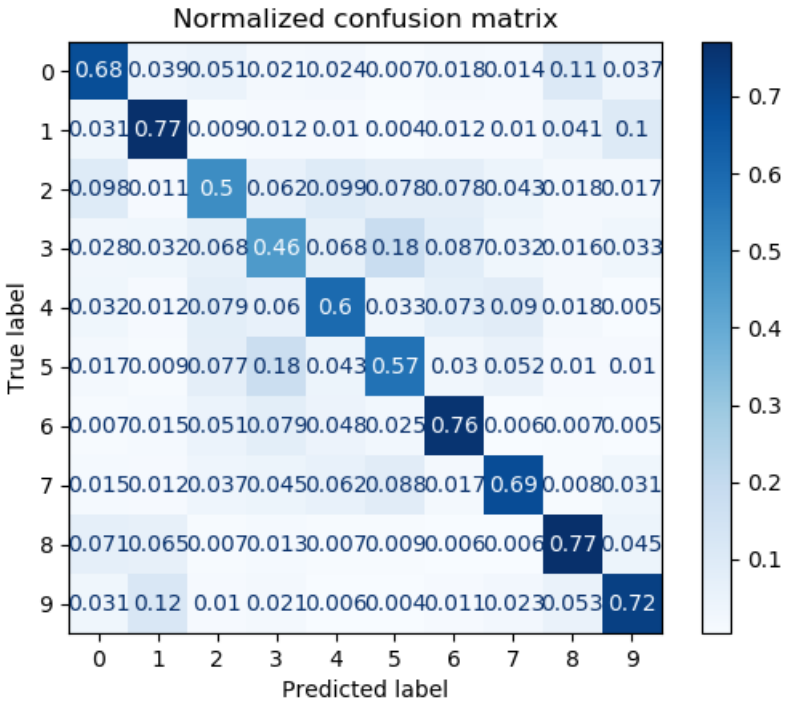

Fig. 3.2: The normalized confusion matrix
when the number of train images in Module2&3 is 50k by the Bin method,
where airplane 0, automobile 1, bird 2, cat 3, deer 4, dog 5, frog 6, horse 7, ship 8, and truck 9

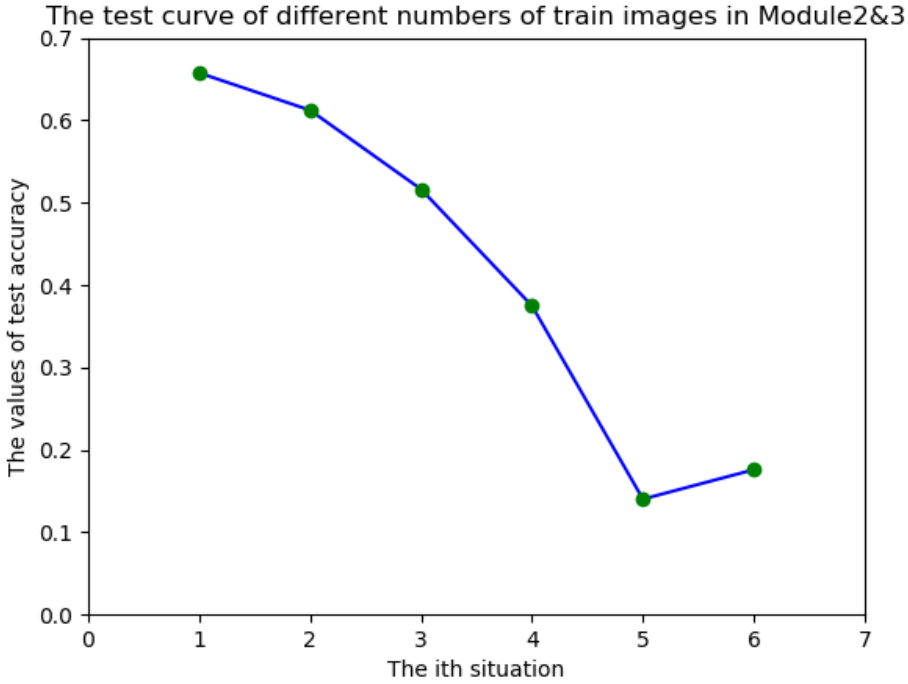

Fig. 3.3: The test curve of different numbers of train images in Module2&3 from the Table 2.1,
where the x axis 1, 2, 3, 4, 5, 6 represent 1, 1/2, 1/4, 1/8, 1/16, and 1/32 of the train images

3.4 Running time

After using the bin method instead of K-means, the running time especially for the feature extraction goes down at a large scale as shown in the Table 3.5 and the Table 3.6. From the Table 3.7, we could see that after increasing the number of features, the total running time will go up a bit presented in the Table 3.7. The mean-pooling way should be discarded without doubt because not only it will increase running time, but also it will decrease the test accuracy.

Table 3.5: The running time of the PixelHop++ model of different numbers of training samples in Module2&3 with K-means in feature selection

	All	1/4	1/8	1/16	1/32
PixelHop fitting	401.71s	401.71s	401.71s	401.71s	401.71s
PixelHop transform	4960.04s	259.88	188.71s	157.56s	137s
Max pooling	220.85s	201.57	202.03	202.7s	206.22s
Feature extraction	2857.50s	1183.24	937.55	786.36s	790.93s
LAG part	114.6s	15.28	9.41	6.05s	4.23s
Random forest part	80.98s	20.11	10.39	4.73s	0.89s
Total	8635.68s	2081.79s	1749.8s	1559.11s	1540.98s
Use AWS?	No	Yes	Yes	Yes	Yes

Table 3.6: The running time of the PixelHop++ model of different numbers of training samples in Module2&3 with the Bin method in feature selection

	All	1/4	1/8	1/16	1/32
PixelHop fitting	401.71s	401.71s	401.71s	401.71s	401.71s
PixelHop transform	5128.68s	305.16s	220.84s	177.98s	156.87s
Max pooling	228.15s	221.53s	213.30s	209.57s	202.53s
Feature extraction	334.91s	29.24s	17.30s	11.16s	8.30s
LAG part	117.86s	29.18s	15.94s	8.36s	5.84s
Random forest part	87.63s	19.53s	9.06s	3.71s	0.85s
Total	6298.94s	1006.35	878.15s	808.78s	776.1s
Use AWS?	No	No	No	No	No

Table 3.7: The running time of the PixelHop++ model of different numbers of training samples in Module2&3 with the Bin method in feature selection in three situations based on the parameter-setting of the Table 3.6

	Features: 3000, 2000, and 275	Min-pooling	Mean-pooling
PixelHop fitting	401.71s	401.71s	401.71s
PixelHop transform	5633.82s	5605.43s	5538.08s
Max pooling	225.31s	224.69s	1380.86s
Feature extraction	313.20s	323.35s	335.32s
LAG part	190.11s	122.42s	114.43s
Random forest part	88.83s	88.17s	79.57s
Total	6852.98s	6765.77s	7849.97s
Use AWS?	No	No	No

3.5 Model size

The procedure of getting my model size is shown below:

According to the TA's discussion, I can approximate my model size by figure out the total parameter numbers in three parts.

First of all, I have to find out the number of Saab filter coefficients in the PixelHop++ units. In this part, with the TA's hint, the number of parameters of the first PixelHop unit is

$$5 \times 5 \times 3 \times 42 = 3150,$$

the number of parameters of the second PixelHop unit is

$$5 \times 5 \times 274 = 6850,$$

and the number of parameters of the third PixelHop unit is

$$5 \times 5 \times 529 = 13225,$$

where 5 means the size of the neighborhood, 3 means the number of the image's channel, and 42, 274, and 529 are the numbers of spectrum after the max-pooling part for each PixelHop unit.

Secondly, I have to find the number of coefficients of the regression matrix in the LAG units. With the suggestion of the TA, the parameter of the regression matrix can be viewed as

$$M \times (n + 1),$$

where n is the feature dimension and the M is the output dimension of LAG. In my model, n_1, n_2, n_3 are 1750, 1000, 250 respectively and , M_1, M_2, M_3 are all 50.

Lastly, I have to find the number of parameters of the classifier. I have used the random forest algorithm, during which I have set *n_estimators* as 100, to carry out the feature selection, which is very hard to figure out its parameters. Therefore, according to the TA's advice, I can approximate my classifier as a linear regression classifier. In this way, my model has 3 hops and 10 classes. What's more, the output dimension M of LAG is 50.

Therefore, the size of my model is

$$(3150 + 6850 + 13225) + 50 \times (1750 + 1 + 1000 + 1 + 250 + 1) + (3 \times 50 \times 10) = 174{,}875$$

When I use the combination (3000, 2000, 275), the size of my model will become

$$(3150 + 6850 + 13225) + 50 \times (3000 + 1 + 2000 + 1 + 275 + 1) + (3 \times 50 \times 10) = 288{,}625$$

In contrast, the model size in Problem2 of HW5 is 15,001,418, which is tremendously bigger.

3.6 Summary

According to my experiments, I have three seemingly good results as shown in the Table 3.8.

Table 3.8: The summary three results

	K-means with (1750, 1000, 250)	Bin method with (1750, 1000, 250)	Bin method with (3000, 2000, 275)
Accuracy	0.6512	0.6311	0.6575
Running time	8635.68s	6298.94s	6852.98s
Model size	174875	174875	288625

References

[1] C-C Jay Kuo, Min Zhang, Siyang Li, Jiali Duan, and Yueru Chen, “Interpretable convolutional neural networks via feedforward design,” *Journal of Visual Communication and Image Representation*, vol.60, pp. 346–359, 2019.
[2] Yueru Chen and C-C Jay Kuo, “Pixelhop: A successive subspace learning (ssl) method for object recognition,” *Journal of Visual Communication and Image Representation*, p. 102749, 2020.
[3] Yueru Chen, Mozhdeh Rouhsedaghat, Suya You, Raghuveer Rao, C.-C. Jay Kuo, “PixelHop++: A Small Successive-Subspace-Learning-Based (SSL-based) Model for Image Classification,” *https://arxiv.org/abs/2002.03141*, 2020
[4] [Online] Available: https://github.com/USC-MCL/EE569_2020Spring
[5] [Online] Available: https://scikitlearn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py