EE569 Digital Image Processing

HOMEWORK #6

Issued: 04/08/2020
Problem1 due: 04/26/2020
Problem2 due: 05/03/2020

Yao Fu
6786354176
yaof@usc.edu

EE 569 Homework #6
April 26, 2020

Problem 1: Understanding Successive Subspace Learning (SSL) (50%)

(a) Feedforward-designed Convolutional Neural Networks (FF-CNNs) (20%)

(b) Successive Subspace Learning (SSL) (30%)

1.1 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.

In order to understand the basic mechanism of feedforward-designed convolutional neural networks and successive subspace learning, in this report, I will read the papers [1], [2], and [3] to answer required questions. Necessary results and images will be presented later in my report.

1.2 Approach and Procedures

(a) Feedforward-designed Convolutional Neural Networks (FF-CNNs)

Question (1):

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.

The computational neuron, which usually consists of two stages: affine computation and nonlinear activation, is frequently regarded as a basic building element of artificial neural networks. The input is an N-dimensional random vector $x = (x_0, x_1, \dots, x_{N-1})^T$. The kth neuron has N filter weights that can be represented in a vector form as $a_k = (a_{k,0}, a_{k,1}, \dots, a_{k,N-1})^T$, and one bias term b_k. The affine computation is

$$y_k = \sum_{n=0}^{N-1} a_{k,n} x_n + b_k = a_k^T x + b_k, \quad k = 0,1,\dots,K-1, \tag{1}$$

where a_k is the filter weight vector associated with the kth neuron. With the ReLU nonlinear activation function, the output can be written as

$$z_k = \emptyset(y_k) = \max(0, y_k). \tag{2}$$

Hence, with the background information above, it's not hard to reach a point that 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. 1.1 is a simple flow diagram that can display the process of the Saab transform, which can replace the convolutional layers of the LeNet-5.

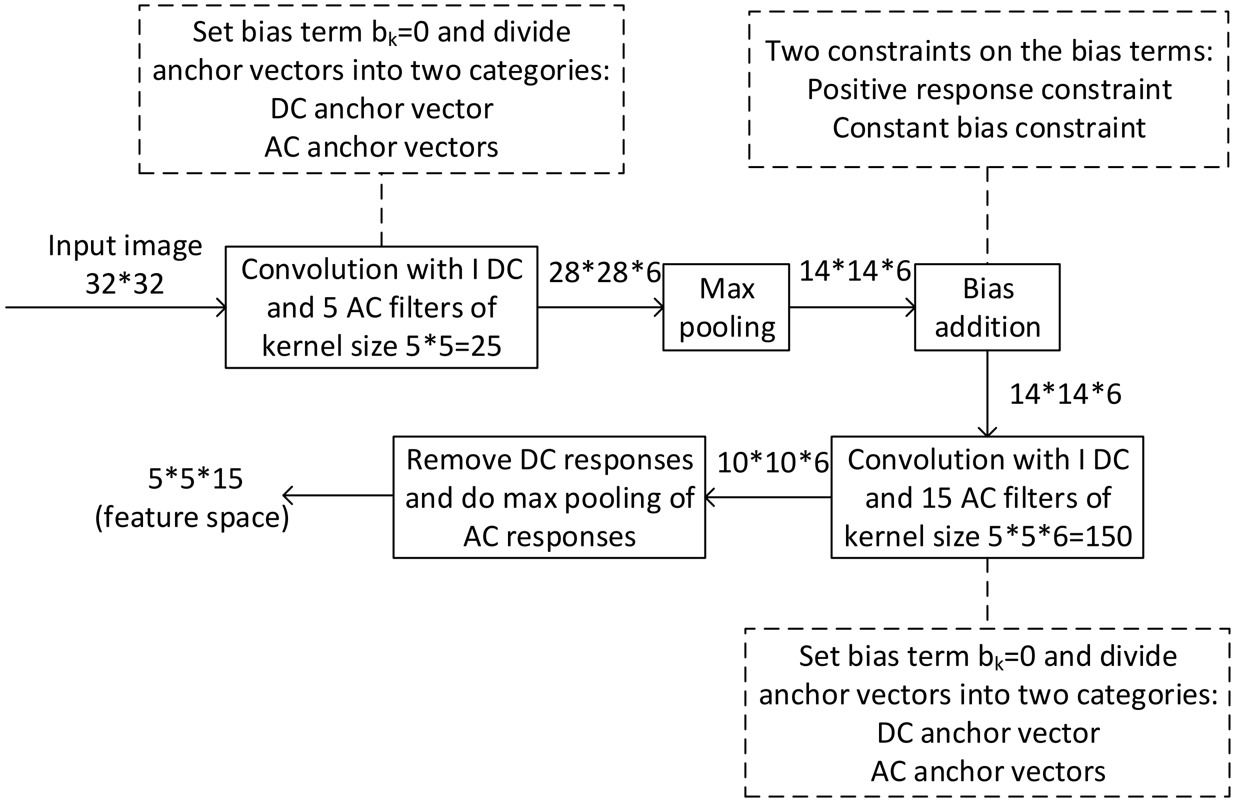

Set bias term b_k=0 and divide anchor vectors into two categories: DC anchor vector AC anchor vectors
Two constraints on the bias terms: Positive response constraint Constant bias constraint
Input image 32*32
Convolution with I DC and 5 AC filters of kernel size 5*5=25
28*28*6
Max pooling
14*14*6
Bias addition
14*14*6
Convolution with I DC and 15 AC filters of kernel size 5*5*6=150
10*10*6
Remove DC responses and do max pooling of AC responses
5*5*15 (feature space)
Set bias term b_k=0 and divide anchor vectors into two categories: DC anchor vector AC anchor vectors

Fig. 1.1: 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 $a_k, k = 1, \dots, K-1$.

Question (2):

The similarities and differences between feedforward-CNNs and backpropagation-designed CNNs can be stated as follows:

Similarities

(1) Although the BP design will utilize fully connected layers to make decisions while the FF design will utilize LSR operations, yet they both need data labels, which means that they both belongs to supervised learning.
(2) Both models in terms of two approaches are not robust enough. Specifically speaking, they are both vulnerable to disturbance, which leads to poor performance curves.
(3) The idea of the BP design is completely derived from the LeNet-5, which is a simple convolutional neural network. From my point of view, the whole architectures of two methods are very similar, indicating we can simply regard them as the combination of "feature extraction" and "decision making".
(4) Both models' inner mechanism, to some extent, can be explainable with respect to the field of digital signal processing.

Differences

(1) The BP design focuses utilizing a group of training and testing data, a certain network architecture and a selected cost function to train a model by applying an optimization technique. In contrast, the FF design tries to exploit data statistics to figure out spatial-spectral transformations in convolutional layers, where no data labels are needed. Also, after that, the BP design will utilize fully connected layers to make decisions while the FF design will utilize LSR operations.
(2) The BP design usually relies on some operation techniques such as stochastic gradient descent in order to minimize the cost function. By comparison, the FF design basically relies on linear algebra.
(3) Even if the interpretability of CNNs based on BP has been studied in some papers, it's still very challenging to be totally interpreted. However, CNNs based on FF design are mathematically transparent to individuals.
(4) The whole neural network based on BP is end-to-end indispensably coupled, which tightly depends on input data and output labels. On the contrary, the FF design can decouple the whole network into two modules(the "feature extraction" one and the "classification" one) explicitly. Moreover, the FC layers of the FF design can be replaced with the random forest and support vector machine classifiers while the BP design only accepts the fully connected layers to make decisions.
(5) It usually takes a long time for the network based on BP is to converge while the FF design is significantly faster according to the authors' experiments.
(6) As for the BP design, it's inevitable that we have to design different neural networks for different tasks ranging from image segmentation to object tracking. In contrast, thanks to the FF design, we can apply the same convolutional layers to handle all tasks and merely design different FC layers.

(b) Successive Subspace Learning (SSL)

Question (1):

Broadly speaking, the purpose of subspace learning is to discover subspace models conductive to realizing concise data representation, which indicates discarding less important features to retain relatively dominant features, and implementing accurate decision making based on training data with existing machine learning algorithms. When we consider subspace learning in terms of multiple stages, the terminology called "Successive Subspace Learning" appears, which has some similar meaning with the convolutional neural networks whose response in a deeper layer usually has a larger receptive field.

From the paper[2], the process of successive subspace learning normally consists of four steps:

Step1: We will successively expand a target pixel's neighborhood based on the near-to-far concept in multiple stages;

Step2: We will utilize subspace approximation to carry out unsupervised dimension reduction at each stage;

Step3: We will utilize a novel method named label-assisted regression to carry out supervised dimension reduction;

Step4: We will concatenate features from each subspace to form a composite feature set and utilize machine learning algorithms ranging from random forest to support vector machine to make decisions.

The similarities and differences between Deep Learning and Successive Subspace Learning can be stated as follows:

Similarities

(1) They both want to collect attributes of the input images from local to global by employing successively growing a pixel's neighborhoods.

(2) They both use the idea of convolutional filters to regard spatial-domain patterns as spectral information.

(3) They both adopt spatial pooling such as max pooling in order to reduce redundancy.

Differences

(1) **Model expandability**: DL is a parametric model that is usually of a very large model size, thus leading to a waste of resource easily. By comparison, SSL adopts a model without parameters, indicating that the model is flexible to add and/or delete filters at several units so that people can adjust model complexity easily.

(2) **Incremental learning**: It's almost impossible for a trained DL model to handle new data classes. However, we can add more Saab filters or expand the regression matrix to enable SSL to handle new classes.

(3) **Model architecture**: For DL, we usually use backpropagation to determine parameters by defining a cost function. In contrast, for SSL, we could extract features from PixelHop units to conduct ensemble learning on them.

(4) **Model interpretability**: The DL model is a black box tool that is very hard to be understood while the SSL model is a white box whose mathematical mechanism is accessible.

(5) **Model parameter search**: DL models' parameters are determined by using backpropagation with a cost function. By comparison, the SSL's pipeline is based on the idea of feedforward design.
(6) **Training complexity**: DL models frequently demand a great many computing resources in order to finish training while the training complexity of SSL is tremendously lower.
(7) **Spectral dimension reduction**: People use convolutions in DL to transform one representation to another while convolutions in SSL are aimed at getting projections onto principal components of the certain subspace.
(8) **Task-independent features**: The derived features from DL are task dependent because parameters of models have to be determined by both input images and output labels. On the contrary, the features from unsupervised dimension reduction are task-independent while ones from supervised dimension reduction are task-dependent.
(9) **Multi-tasking**: DL models' joint cost function integrated by several tasks may not be optimal to each task. Conversely, SSL is able to acquire task-independent features and feed them into different LAG units and classifiers, making it possible to achieve multi-tasking.
(10) **Incorporation of priors and constraints**: DL may add new terms to the original cost function, whose impact is usually trivial to the learning process. SSL can use priors and constraints to prune attributes of near-to-far neighborhoods.
(11) **Weak supervision**: A large number of labeled data are needed to train DL models. However, for SSL, the unsupervised dimension reduction process doesn't necessarily demand labels. Labels are only needed in the LAG units and the training of a classifier.
(12) **Adversarial attacks**: DL networks are easily influenced by a small perturbation while in SS the small perturbation can be easily filtered out by PCA, which can make SSL models more robust.

Question (2):

The Fig. 1.2 is a simple framework of Successive Subspace Learning, where the functions of three modules are briefly stated.

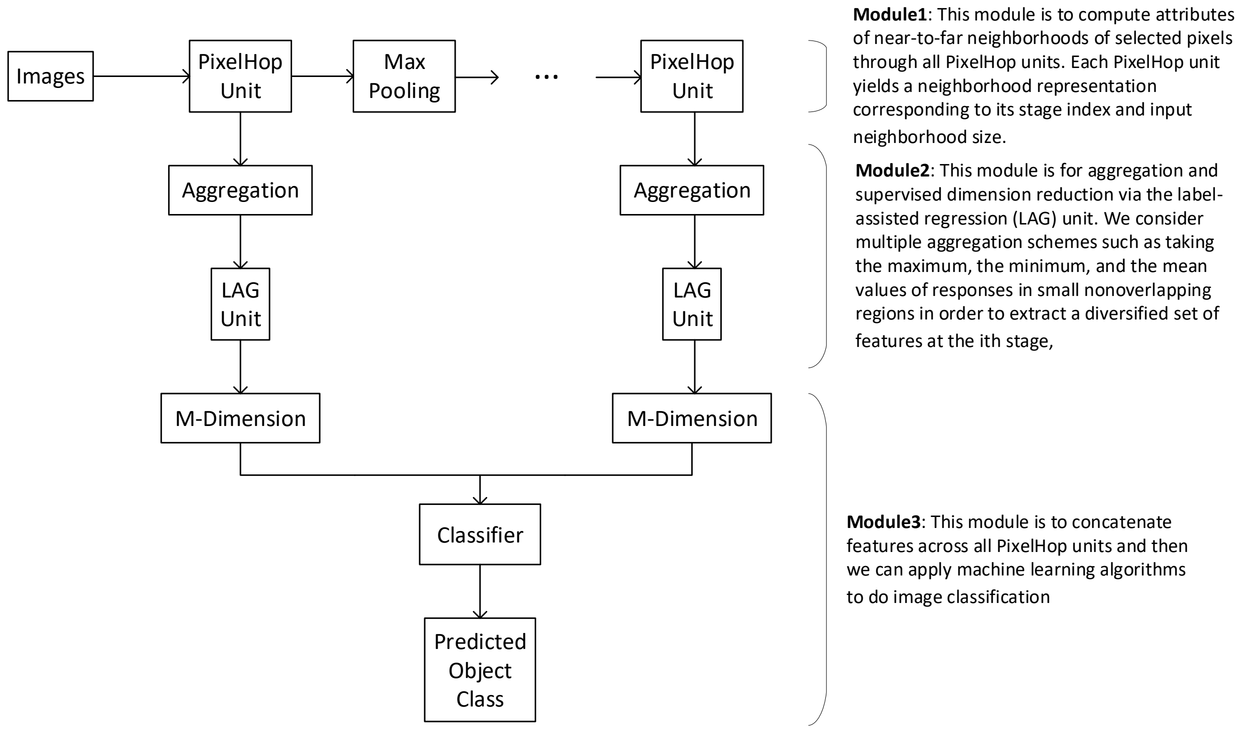

Images
PixelHop Unit
Max Pooling
…
PixelHop Unit
Aggregation
Aggregation
LAG Unit
LAG Unit
M-Dimension
M-Dimension
Classifier
Predicted Object Class
Module1: This module is to compute attributes of near-to-far neighborhoods of selected pixels through all PixelHop units. Each PixelHop unit yields a neighborhood representation corresponding to its stage index and input neighborhood size.
Module2: This module is for aggregation and supervised dimension reduction via the label-assisted regression (LAG) unit. We consider multiple aggregation schemes such as taking the maximum, the minimum, and the mean values of responses in small nonoverlapping regions in order to extract a diversified set of features at the ith stage,
Module3: This module is to concatenate features across all PixelHop units and then we can apply machine learning algorithms to do image classification

Fig. 1.2: The framework of SSL and the functions of three modules

Question (3):

The PixelHop Unit

The process of neighborhood construction and subspace approximation steps in the PixelHop unit is briefly stated blow:

For the ith PointHop unit in the module1 when $i = 1, \ldots, I$, it concatenates attributes of the $(i-1)$th neighborhood of a target pixel and its $(S_{i-1} \times S_{i-1} - 1)$ neighboring pixels. We can denote dimension of the enlarged neighborhood as

$$S_{i-1} \times S_{i-1} \times K_{i-1},$$

where $K_{i-1}$is the dimension of the ith PixelHop unit. It is very necessary to adopt a technique in order to maintain the rapidly growing dimension in a controllable size, which is achieved by the Saab transform. The simple flowchart of the procedure is shown in the Fig. 1.3.

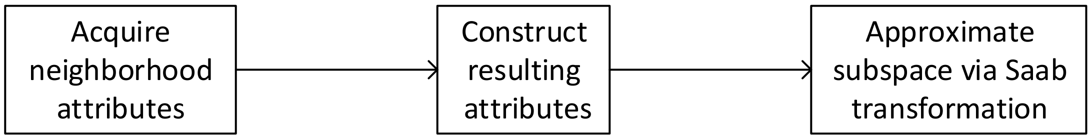

Acquire neighborhood attributes
Construct resulting attributes
Approximate subspace via Saab transformation

Fig. 1.3: The framework of each PixelHop Unit

Each PixelHop unit presents a certain type of neighborhood features with respect to its stage index and input size. At the ith PixelHop unit, we utilize the Saab transform technique to reduce the spectral dimension while the spatial dimension just remains the original size. In order to handle the spatial redundancy due to the overlapping effect at each PixelHop unit, we can apply maximum pooling layer between two consecutive PixelHop units.

The PixelHop++ Unit

Generally speaking, the PixelHop++ method is an improved version of PixelHop, which means that they share lots of similarity with each other. They are both composed of three parallel modules and apply the Label-Assisted Regression (LAG), which will be discussed later. As for their processes of neighborhood construction and subspace approximation steps, both of them have the same goal of realizing successive near-to-far neighborhood expansion and unsupervised dimension reduction but the details of implementation have some difference, meaning that the PixelHop++ Unit has the modifications below:
(1) The traditional Saab transform is replaced by the channel-wise (c/w) Saab transform in the PixelHop++ method.
(2) The authors in the paper[3] design a novel tree-decomposed feature representation.
(3) The leaf node's features are ordered in terms of their cross-entropy values, by which a feature subset can be selected.

Specifically, the comparison between traditional Saab transform and the c/w Saab transform is shown in the Fig. 1.4. 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.

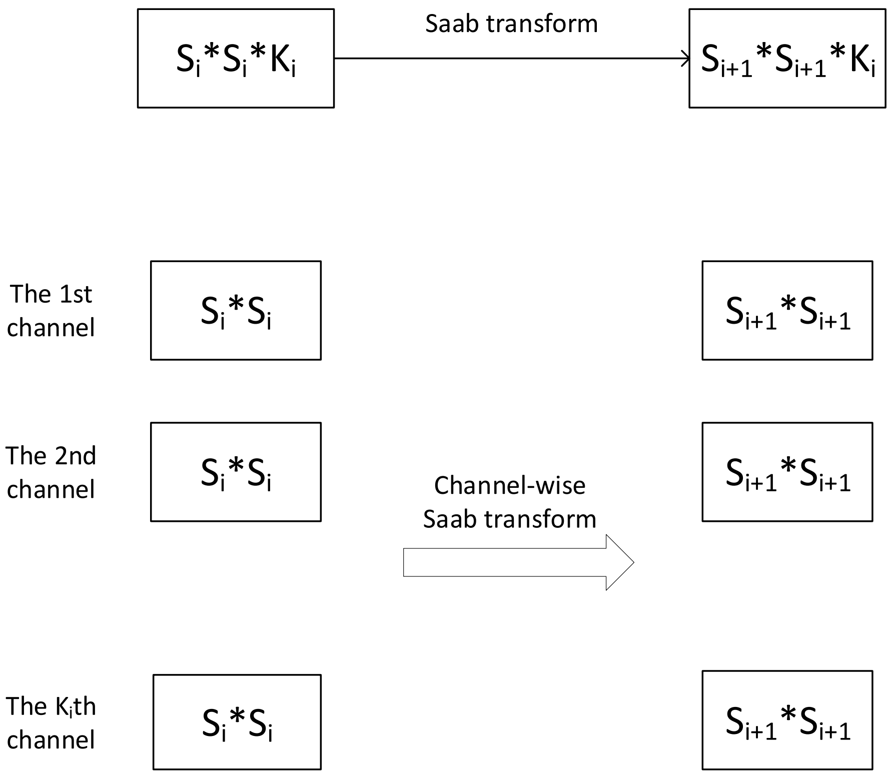


Fig. 1.4: The comparison between
the traditional Saab transform and the c/w Saab transform

Question (4):

Role: Supervised dimension reduction.

We have to integrate the local-to-global representations or attributes provided by multiple PixelHop units in order to achieve the image classification. What's more, in order to mimic convolutional neural networks' using data labels effectively via backpropagation, discovering an approach to using data labels in Successive Subspace Learning is desirable, thus resulting in the design of Label-Assisted Regression.

Procedure:

Step1: Apply K-means algorithm to cluster samples belonging to the same class or category to create object-oriented subspaces and find out the relevant centroids of each cluster.
Step2: Calculate the target probability vector in terms of Euclidean distance.

Let x_j denote a training sample whose class is j and $c_{j,l}$ denote the lth centroid who belongs to the class j. So, we could use the formula below to get the target probability vector.

$$\begin{cases} Prob\left(x_j, c_{j',l}\right) = 0, \quad if\ j \neq j' \\ Prob\left(x_j, c_{j,l}\right) = \dfrac{\exp\left(-ad(x_j, c_{j,l})\right)}{\sum_{l=1}^{L} \exp\left(-ad(x_j, c_{j,l})\right)}, \quad if\ j = j' \end{cases}$$

If x_1 belongs to the class1, we can use the formula below to compute the target probability vector.

$$Target(x_1) = \begin{bmatrix} P_1(x_1) \\ P_2(x_1) \end{bmatrix} = \begin{bmatrix} Prob\left(x_1, c_{1,1}\right) \\ Prob\left(x_1, c_{1,2}\right) \\ Prob\left(x_1, c_{1,3}\right) \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Step3: Once we know a feature vector x from a training sample and its pertinent target probability vector space, we can learn a regression matrix which is shown blow:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & w_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & w_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{Mn} & w_M \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \begin{bmatrix} p_1(x) \\ \vdots \\ p_j(x) \\ \vdots \\ p_J(x) \end{bmatrix}.$$

Step4: Once acquiring the learnt regression matrix, we could use it to get the predicted probability vector for a test sample.

Advantages:

(1) The traditional fully connected layers will use backpropagation to determine weight coefficients, which is very slow. Using the LAG can significantly increase the computation speed.
(2) Different from fully connected layers, the process of LAG is transparent and mathematically explainable.
(3) The predicted probability has higher discriminant power than the original feature after using an interpretable method while we have no idea about how to increase discriminant power when

using fully connected layers.

Problem 2: CIFAR-10 Classification using SSL (50%)

(a) Building a PixelHop++ Model (35%)

(b) Error analysis (15%)

2.1 Abstract and Motivation

The fundamental mechanism and advantages of Successive Subspace Learning have already been clearly talked about in the previous section. In the Problem2, I will build a simple PixelHop++ Model with the TA's help and do error analysis in order to further understand the practical implementation of SSL.

2.2 Approach and Procedures

(a) Building a PixelHop++ Model

In this part, 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.

Then, with the saved PixelHop model, I will train Module2&3 on all the training images. The training time, training accuracy, and model size will be discussed in the section 2.4.

Next, I am going to apply my model to 10k testing images and show the resulting test accuracy.

Finally, with the saved model from the module1 unchanged, I will make the number of training samples in the Module2&3 1/2, 1/4, 1/8, 1/16, and 1/32 of the initial 50k. The test curves and results will be presented in the section 2.4.

(b) Error analysis

In this part, I will use Python3 and codes from [5] to acquire the confusion matrix and represent it as a heat map based on the results when the number of training images in Module2&3 is 25k and 50k. The results will be discussed later.

2.3 Experimental Results

Table 2.1: The training time and test accuracy of the PixelHop++ model of different numbers of training samples in Module2&3 with the instance type "m5d.8xlarge" from AWS or my own laptop

	All	1/2	1/4	1/8	1/16	1/32
PixelHop fitting	401.71s	401.71s	401.71s	401.71s	401.71s	401.71s
PixelHop transform	4960.04s	409.25	259.88	188.71s	157.56s	137s
Max pooling	220.85s	212.85	201.57	202.03	202.7s	206.22s
Feature extraction	2857.50s	1705.26	1183.24	937.55	786.36s	790.93s
LAG part	114.6s	23.33	15.28	9.41	6.05s	4.23s
Random forest part	80.98s	28.3304	20.11	10.39	4.73s	0.89s
Total	8635.68s	2780.7304s	2081.79s	1749.8s	1559.11s	1540.98s
Use AWS?	No	Yes	Yes	Yes	Yes	Yes
Test accuracy	0.6512	0.6281	0.5915	0.5205	0.4028	0.1302

When the module2 and module3 have 50k training images, the total running time with my laptop whose CPU is Intel Core-i7-10th-Generation is 8635.68s and train accuracy is 1.0. When applying the trained model to 10k testing images, the test accuracy is 0.6512.

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 second 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. At first, I fail to extract 1000 features in the PixelHop3 so that I use the combination (1750, 1000, 250). Another way is to choose top 50% of the features, which is also good, but I don't have to try it according to the TA's email due to different requirements shown in the discussion slides and the homework requirement.

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

$$(3150 + 6850 + 13225) + 50 \times (1750 + 1 + 1000 + 1 + 250 + 1) + (3 \times 50 \times 10) = 174875$$

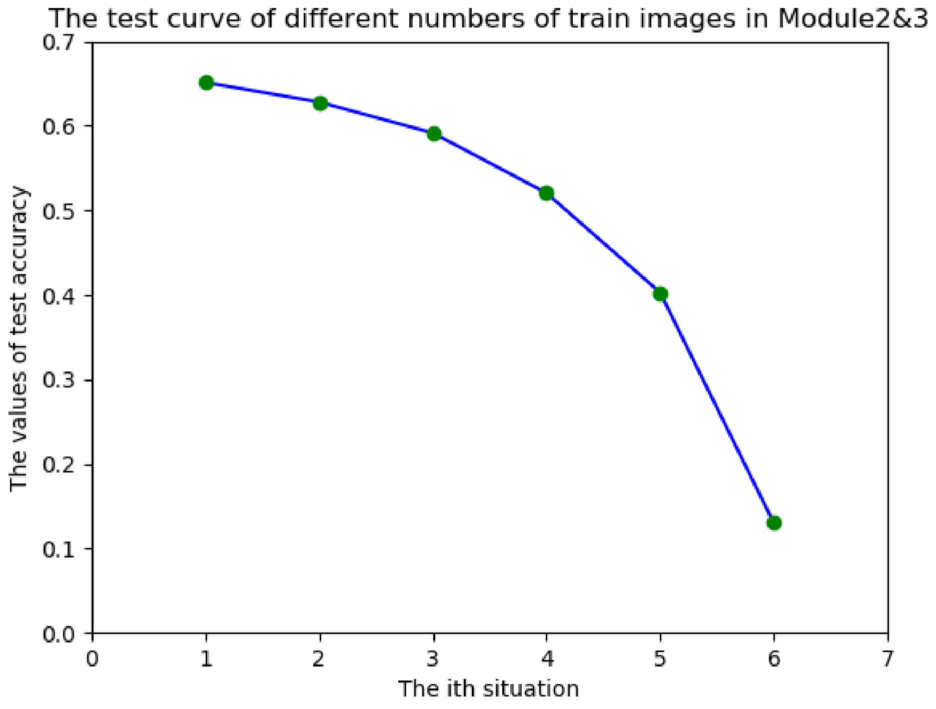
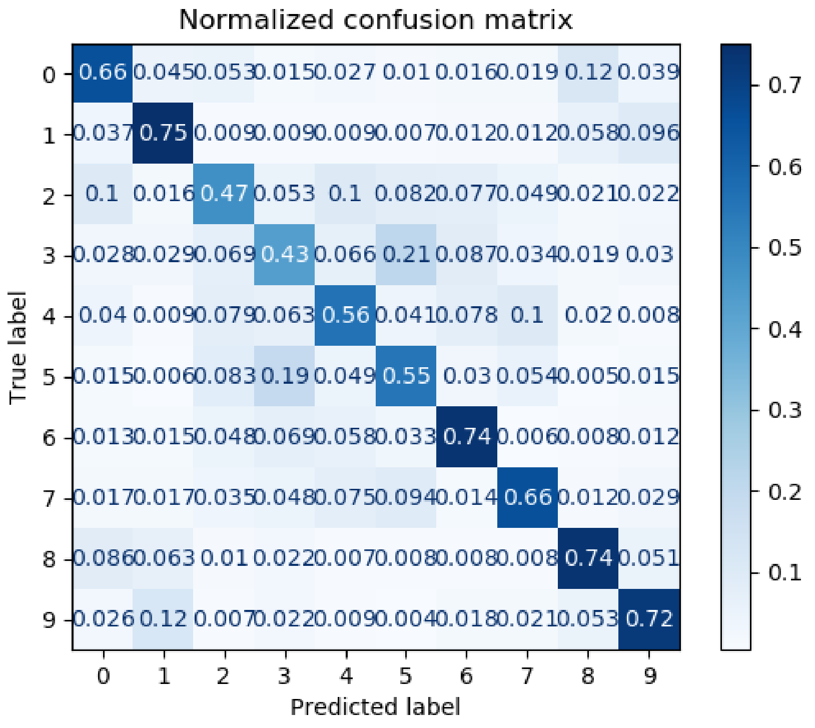


Fig. 2.1: The test curve of different numbers of train images in Module2&3 from the Table 2.1, where the x axis 1, 2, 3, 4, 5, 6 represent 1, 1/2, 1/4, 1/8, 1/16, and 1/32 of the train images



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	0.66	0.045	0.053	0.015	0.027	0.01	0.016	0.019	0.12	0.039
1	0.037	0.75	0.009	0.009	0.009	0.007	0.012	0.012	0.058	0.096
2	0.1	0.016	0.47	0.053	0.1	0.082	0.077	0.049	0.021	0.022
3	0.028	0.029	0.069	0.43	0.066	0.21	0.087	0.034	0.019	0.03
4	0.04	0.009	0.079	0.063	0.56	0.041	0.078	0.1	0.02	0.008
5	0.015	0.006	0.083	0.19	0.049	0.55	0.03	0.054	0.005	0.015
6	0.013	0.015	0.048	0.069	0.058	0.033	0.74	0.006	0.008	0.012
7	0.017	0.017	0.035	0.048	0.075	0.094	0.014	0.66	0.012	0.029
8	0.086	0.063	0.01	0.022	0.007	0.008	0.008	0.008	0.74	0.051
9	0.026	0.12	0.007	0.022	0.009	0.004	0.018	0.021	0.053	0.72

Fig. 2.2: The normalized confusion matrix when the number of train images in Module2&3 is 25k, where airplane 0, automobile 1, bird 2, cat 3, deer 4, dog 5, frog 6, horse 7, ship 8, and truck 9

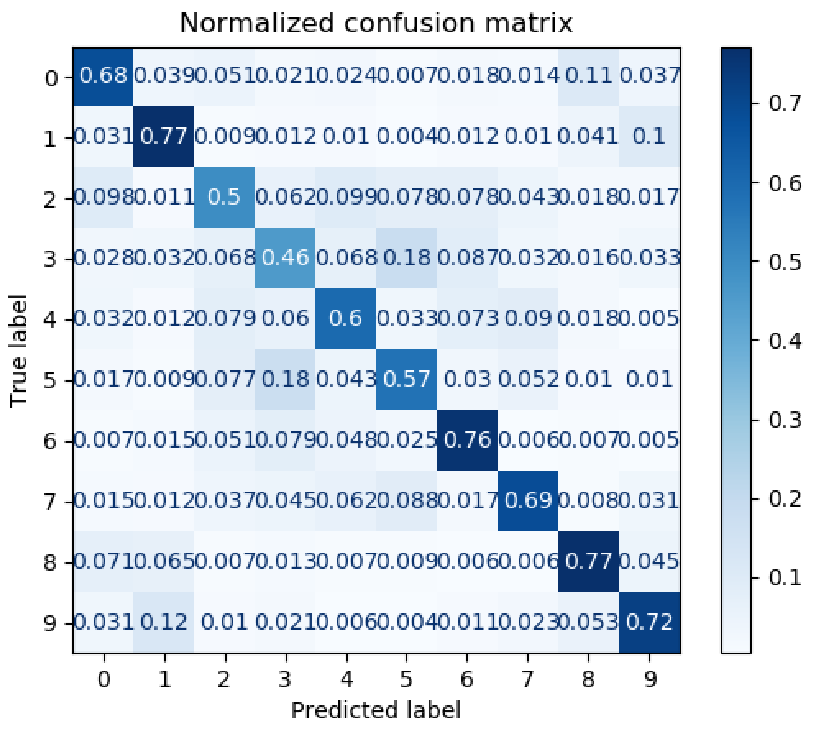


Fig. 2.3: The normalized confusion matrix when the number of train images in Module2&3 is 50k, where airplane 0, automobile 1, bird 2, cat 3, deer 4, dog 5, frog 6, horse 7, ship 8, and truck 9

2.4 Discussion

My discussion of experiment results of Problem 2 is stated as below.

In part (a), when applying 50k training images to train Module2&3, the total training time is 8635.68s. One thing needing to be mentioned is that at first I didn't know to use batch processing to save memory in order to enable the program to run on my own laptop. In that case, I have utilized the AWS's instance type "m5d.8xlarge" to handle situations when training samples are 1/2, 1/4, 1/8, 1/16, and 1/32 of the total. However, finally with my own laptop, whose memory is 16GB and CPU is Intel-Core-i7-10-Genneration, I utilize the batch method to successfully run the program when the number of training samples in Module2&3 is 50k. Moreover, the train accuracy is 1.0, indicating that it's overfitting and I can modify the depth of decision trees to handle it, and the test accuracy is 0.6512. What's more, the model size is 174875, whose process of calculation is clearly stated in the section 2.3. Lastly, the test curve of different numbers of train images in Module2&3 is shown in the Fig. 2.1, which clears shows that when the numbers of train imaged decrease, the accuracy drops significantly. Hence, in order to achieve high accuracy, we have try to use more train images but when the number is high enough to reach the bottleneck we have to find other ways to increase test accuracy.

In part(b), I use the online source from [5] to draw two heat maps of the resulting confusion matrices, which are shown in the Fig. 2.2 and the Fig. 2.3. For the Fid. 2.3 when the number of train images is 50k, it's not hard for us to see that the "cat" class is the most difficult one and both the “automobile” and the “ship” classes yield the lowest error rate. From the Fig. 2.2, the “automobile” has the lowest error rate. Moreover, the confusing group is “cat” and “dog”. From my point of view, they are animals having a similar shape of bodies just as shown in the Fig. 2.4, making it hard for computers to discriminate them, especially when there are some disturbances ranging from occlusion to noise. Lastly, I believe we can increase the number of PixelHop units to extract more discriminant features to distinguish difficult classes. Also, in the Module1 I have only used 10k training images due to the memory limit, there is still lots of room to improve accuracy.

Fig. 2.3: A cat and a dog

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