

EE 569

Homework #6:

**Understanding Successive Subspace Learning
and CIFAR-10 Classification using SSL**

Name: Zhiwei Deng

Date: 4/20/2020

Contents

1. PROBLEM 1.....	3
1.1 FEEDFORWARD-DESIGNED CONVOLUTIONAL NEURAL NETWORKS (FF-CNNs).....	3
1.1.1 Saab Transform Summary.....	3
1.1.2 Comparison between FF-CNN and BP-CNN.....	4
1.2 SUCCESSIVE SUBSPACE LEARNING (SSL).....	6
1.2.1 SSL Methodology and Comparison with DL.....	6
1.2.2 Modules Functions.....	9
1.2.3 Neighborhood Construction and Subspace Approximation.....	10
1.2.4 Label-Assisted Regression (LAG).....	13
2. PROBLEM 2.....	15
2.1 BUILDING A PIXELHOP++ MODEL.....	15
2.1.1 Abstract and Motivation.....	15
2.1.2 Approach and Procedures.....	16
2.1.3 Experimental Result.....	18
2.1.4 Discussion.....	21
2.2 ERROR ANALYSIS.....	25
2.2.1 Abstract and Motivation.....	25
2.2.2 Approach and Procedures.....	25
2.2.3 Experimental Result.....	26
2.2.4 Discussion.....	26
REFERENCES.....	31

1. Problem 1

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

1.1.1 Saab Transform Summary

The flow diagram of the Saab transform is shown as below in Figure 1.

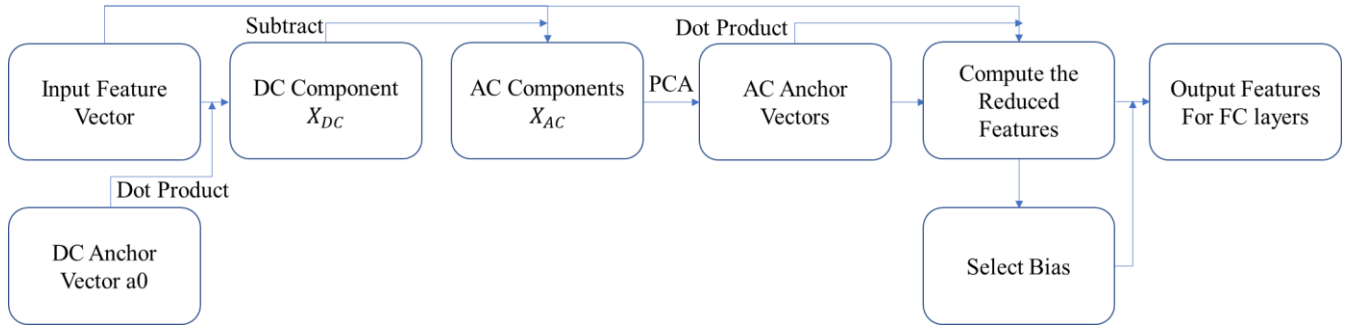


Figure 1. Flow Diagram of Saab Transform

Explanation: The Saab transform's purpose and function are to reduce the dimension of the features that constructed by the input images. The features can be constructed by the sliding window approach. And the outputs of the Saab transform are the dimension reduced features. The procedure of the Saab is stated as below.

Firstly, for each feature vector, we use the normalized all-one vector as the DC anchor vector a_0 . The formula of a_0 is shown as below in equation (1). Then we use this anchor vector to compute the DC component of certain features. The DC component is shown as equation (2). The DC component of certain feature is the mean of the feature values, which gives a large-scale property of the feature in big picture.

Secondly, for each feature vector, we use the DC component to compute the orthogonal AC component of the input vector by subtracting the DC

component from the original input. After we get all possible AC components of the input, we conduct PCA on these vectors and get the first $K-1$ vectors as the AC anchor vectors a_k . These anchor vectors can form a transform matrix that can transport a high dimensional feature in spatial to a low dimensional feature, which acts as feature reduction. The formula is shown as equation (3). At this point, all anchor vector selection is finished.

Thirdly, we need to eliminate the sign confusion. To solve the sign confusion problem, we need to select a bias for all feature values to be non-negative. Two methods of bias selection can be applied, we can apply different bias for each anchor vector or apply a constant to all the reduced features. In the paper, the second one is applied. And the formula is shown as equation (4). At this point, the whole Saab transform procedure is over, the dimension of features is reduced, and the sign confusion is avoided.

1.1.2 Comparison between FF-CNN and BP-CNN

Similarity: There are several properties that share between FF-CNN and BP-CNN. Firstly, they are both data-driven approach, they both need training dataset to fit the model and adjust the parameters in the network. They both have 2 main parts in the network—convolutional layers and fully connected layers. Architectures of FF-CNN and BP-CNN are both fixed, and the architectures are defined by the hyper-parameters. For the feature extraction, they both use a sliding window process to construct the features.

Differences: For FF-CNN, it doesn't need a large dataset for convolutional layers but only needs large-scale labeled image dataset to train

FC layers. For this reason, the computation complexity of FF-CNN is much lower. Also, there is no label in the convolutional layers, which makes the parameters are fewer. High generalization ability, FF-CNN can extract more robust features than CNN like the inversed images can also be recognized. The Saab transform is label-free process. The most important is that there is no BP process in the whole FF-CNN procedure, which improves the efficiency greatly than CNN.

For traditional CNN, there are large dataset needed in both convolutional and FC layers, which cause a large consumption of computation resources. The method works based on the BP process to find the lowest fit location of loss, which means we need run the whole dataset multiple times, which is also a waste of computation and time. Compared with the FF-CNN, the training process is sealed, easy to implement and test. The whole process is label based or supervised, unlike FF-CNN, which only uses labels in the FC layers.

1.2 Successive Subspace Learning (SSL)

1.2.1 SSL Methodology and Comparison with DL

Methodology: The approach of SSL is a high-level concept like CNN, which explains why the procedures and flows work. For SSL, there are 4 main process in it, like convolution, pooling operations in CNN. Firstly, neighborhoods construction should convert the images into tensors in several stages as the inputs of the module. Since the dimension of the inputs should grow exponentially, the following step is a unsupervised dimension reduction process that based on approximating the subspaces of each stage's inputs. At each stage, a supervised dimension reduction follows assistance with the image labels. All information that get from above will be concatenated and treated as features of each stage of each image, and a decision making scheme is introduced like the FC layers of CNN. In this part, SVM or RF can be applied as a classifier to train the model. The whole method of SSL can be represented as below in Figure 2.

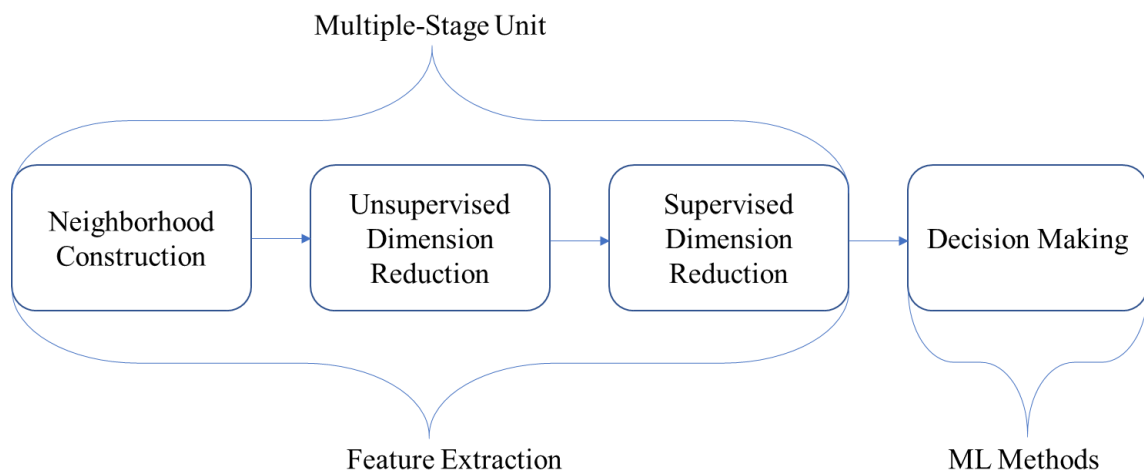


Figure 2. SSL Method Diagram

In a high-level prospect, the SSL method is inspired by the deep learning methodology. According *Kuo.etl*, the first three steps are acting like the convolutional layers in CNN, whose function is extracting the features from the image. The neighborhood construction is collecting the spatial information of the images. But the information contains many redundancies, that's why we need step two. Saab/Saak is an unsupervised process like PCA which can approximate the whole input information using the subspaces while keeping most of its energy. For further extract the useful features and drop those useless ones, a supervised procedure that using the image labels. At this point, all feature extraction process are done. For the last step, SSL just uses the machine learning methods to train the features that extracted above, which can be proved to have similar performances with the FC layers in deep learning.

Comparison: Since the SSL method is inspired by the DL, there are some similarities between each other. However, we can see more differences and changes in them, and SSL gets some significant improvements based on DL in some prospects.

Of course, both two methods are based on data learning and approximation. And for both of SSL and DL, the processing is multi-stage. In CNN, there are multiple convolutional layers and FC layers, which work to extract the features in different receptive fields and making the decision. In SSL, the subspace learning process is cascaded and the input of each stage is the output of the previous stage, like layers in CNN, and for later stages, the

neighborhood construction dimension tends to be bigger. This property is like the CNN's receptive field grows as long as the layers goes deeper. Secondly, to reduce the spatial dimension, both methods apply the pooling methods, like max-pooling in DL and data aggregation in SSL, those methods are applied to reduce the computational complexity and accelerate the algorithm process. Also, the both methods try to get more spectral dimension by processing the spatial information and replace the high dimension growth with slow growth of the spectral dimensions. At last, SSL's process idea is similar with DL, which is that to get features first and then process the features using machine learning techniques.

Although there are some similarities between SSL and DL, they have much more differences in ideas, procedures and methodology. From the big picture, SSL's model architecture has more flexibility than DL, whose architecture can only be Neural Networks. While the architecture of SSL can use different ML techniques and unsupervised learning methods in the 4 steps that described above. Another big differences of them is that the feedforward scheme is applied in SSL but BP scheme is applied in DL, which means to find the best parameters of the model, the SSL spend much less time and computation resources than DL methods. This makes the SSL exceeds the DL methods with lower training and testing complexity. In previous HW, we discussed that the CNN is designed to be task-focused and easy to be attacked by data manipulation. But for SSL, since the feature extraction is absolutely label-free and some dimension reduction is unsupervised, it can detect the

image's task-free features, which make SSL more robust to adversarial attacks. Due to this property, the SSL is more suitable for the multi-task jobs and would be more easily to generalize from one model to another. For the model size, the SSL's model is much smaller than CNN models since there are much fewer parameters in SSL model, which makes the SSL model has much higher scalability than CNNs in mobile devices and edge devices in Internet. Further more, the dimensions of spectral features in DL is fixed as the filter numbers and an unsupervised approximation process is applied in SSL. For the understanding both of them, unlike the DL is a black box, the SSL is much easier to understand since it is pure feedforward procedures in it. At last, for new classes added dataset, CNN needs to be trained from the beginning, while SSL model only needs to add more Saab filters on the existing ones, which shows a higher ability in incremental learning aspect.

1.2.2 Modules Functions

Module 1: Module 1 consists the neighborhood construction and Saab transformation. So, like the convolution layers and max-pooling layers in CNN, module 1 is responsible for extracting informations from the images and reducing the information dimensions in first stage. Firstly, it construct the neighborhoods of certain pixels, with non-loss information and get a neighborhood union. This union's size is proportional to the range of the sliding window's size, so the size of the pixel unions is going to be exponentially explode if we don't reduce its dimension. So, the Saab transform is applied to reduce the information dimension and preserve the

msot of energy of the image features.

Module 2: Module 2's function is to reduce the data dimension in a further manner. Firstly, it uses some aggregation scheme like means and maximums to extract diversified set of features in each stage. After this step, the spatial size of the feature shrinks to P_i from S_i . Then a supervised feature dimension reduction is applied, as the label assisted, the distribution of different classes can be recognized. So, the features' redundant information can be neglected and reduced to form low-dim features of the image.

Module 3: The function of Module 3 is to concatenate the features that extracted by different stages and form a total feature forr certain image. Then it feeds these features to the classifier like SVM or RF to train the classifier as the routine multi-class recognition procedure. In one word, the Module 3 works as a decision making scheme like the FC layer in CNNs.

1.2.3 Neighborhood Construction and Subspace Approximation

PixelHop: For neighborhood construction, as the input tensor of certain PixelHop unit is $K_{i-1} * S_{i-1} * S_{i-1}$, it applies a sliding window scheme to produce a longer tensor by concatenating the pixels in the window with the $K_i * S_i * S_i$, where $K_i = window_size * window_size * K_{i-1}$. Since the procedure is just conctenating the pixels of a window, so the length will be exponentially growing and consume more computation resources if no dimension reduction process appied. The stride may be taken to make sure the sliding windows are not overlapping. The neighborhood construction procedure can be view as the following diagram in Figure 3.

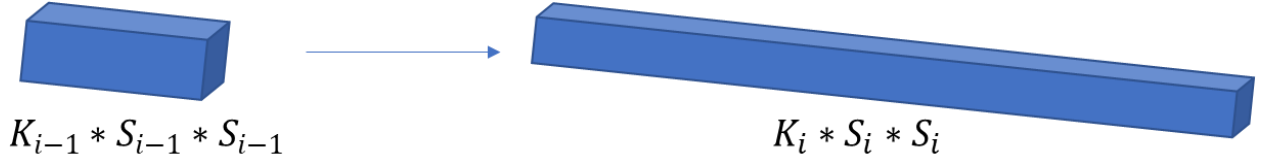


Figure 3. Neighbor Construction

For Saab transform, the steps are stated above in 1.1.1, there are 3 main steps of it. Firstly, it compute the DC components of the input features—mean of the features. Then use it to compute the AC components of input features, where $AC = x - DC$. A PCA process will be applied on the AC components of the features and the most representative subspaces are going to be selected as the approximation of the input spaces. At last, to eliminate the sign confusion, a bias will be added to make each feature values non-negative and the features will be tractable in this mean. It is necessary to mention that in PixelHop, each channel of the input features are going to be processed in a the same manner, which means they are regarded as a whole tensor rather than separate layers.

PixelHop++: For the neighborhood construction, there are few differences between PixelHop and PixelHop++. They both use sliding window manner and concatenate the input features in order as new features, which means the dimensions are also going to explode if no dimension reduction process applied.

For the Saab transform, PixelHop++ uses a new type of Saab transform, which is called channel-wise Saab transform. The basic operation procedures are quiet similar with the traditional Saab transform. There are also three steps—DC components computation, AC components computation

and PCA and the bias addition. However, the c/w Saab transform sees the input data in a different way, rather than regarding the whole tensor as an input, it sees the each channel of the tensor as individual inputs of the transform. And for each channel (layer) it generates multiple channel as the output which can form a longer tensor as the input of the next stage.

Comparison: For neighborhood construction, the procedures of PixelHop and PixelHop++ are quite similar. There are few difference we can discuss about.

However, for the Saab transform and channel-wise Saab transform, we can see the differences between them. In traditional Saab transform, the input is regarded as a whole tensor and computed in the same time. Like the bias selection and the anchor vector selection are all processed together. So, the input's spatial dimension is $\text{window_size} \times \text{window_size} \times K_i$, for instance, if the window size is 3, then the spatial dimension of the input of the traditional Saab transform is $9 \times K_i$. While in c/w Saab transform, the input of the Saab transformation is each single channel of the image tensors. It separates each channel as individual input of Saab transform with the $K_i = 1$, and generate a larger tensor by cascading the output of each channel's output. For instance, if the input's window size is 3, then the size of the input of c/w transform is going to be $3 \times 3 = 9$ rather than $9 \times K_i$.

For channel-wise Saab transform, it should have a improved performance than the traditional Saab transform. Since rather than process the whole tensor in rough. It provides a more detailed way to do the approximation for each

channel. The subspace is now for each channel rather than for the whole tensor. For instance, this approach can avoid the Saab transform to add a very large bias term in every feature space, which might make some feature become less representative. In conclusion, the neighborhood construction procedure of two methods are very similar, but the unsupervised dimension reduction process of PixelHop++ is a little more complicated than the traditional Saab transform in PixelHop, which might produce a better performance.

1.2.4 Label-Assisted Regression (LAG)

The features of each pixel that extracted from the PixelHop unit are called attributes. We can concatenate these attributes to construct the whole features for certain image. But the dimension will be too high to process if we just concatenate them directly. Also, in CNNs, BP uses the image labels effectively. So, the LAG is introduced to do the supervised dimension reduction and use the image label information effectively. The purpose of LAG process is to find the best subspace that formed by the same class samples in the dataset.

There are 3 main steps in LAG. Firstly, LAG clusters the whole samples of same classes into n clusters. These centroids of the clusters can replace the representations of the high dimensional features representation. Then a soft association is applied between each sample and the centroids, the soft association means the output vector represent the probability of the each sub-class. At last, a set of least square regression equations can be set to change

the input vectors to low-dimensional probability vectors. Based on these equations, we can learn the least square regression matrix. And for the test images, the same matrix should also be applied to change the long feature vectors into low-dimensional probability vectors. In conclusion, LAG units reduce the high dimensional representations to low dimensional probability vectors to speed up the classification job and improve the efficiency of the algorithm and it also works like a features filter, to filter out the representative features related to the task based on the labels.

Advantages: Firstly, by using the LAG units, the more redundancies of information in the representations that PixelHop unit extracted are removed. Also, the whole procedure of LAG is feedforward, it saves much computation resources than BP and uses unsupervised techniques(K-means). At last, the LAG unit makes the whole SSL model more flexible. More specifically, if we want to use the same model for a different image processing task, we only need to input the features of PixelHop into different LAG units with different labels, instead of training the whole model like DL, the PixelHop unit and Saab transformation can be totally preserved.

2. Problem 2

2.1 Building a PixelHop++ Model

2.1.1 Abstract and Motivation

The PixelHop++ uses only feedforward design in training process. It uses Saab transformation, label-assisted regression and neighborhood construction with multiple stages. It proves its excellence in training efficiency and interpretability compared with the CNNs.

In this part, I will implement a PixelHop++ model with 3 PixelHop units and apply it on CIFAR-10 datasets to see the training time and testing accuracy. Then I keep the Module 1 unchanged and reduce the labeled training set to different portions of original dataset to see how this reduction of supervision will influence the training accuracy and testing accuracy. The whole architecture of PixelHop++ is shown as Figure 4. The model size, the training accuracy, testing accuracy and training time is going to be discussed at the end of this section.

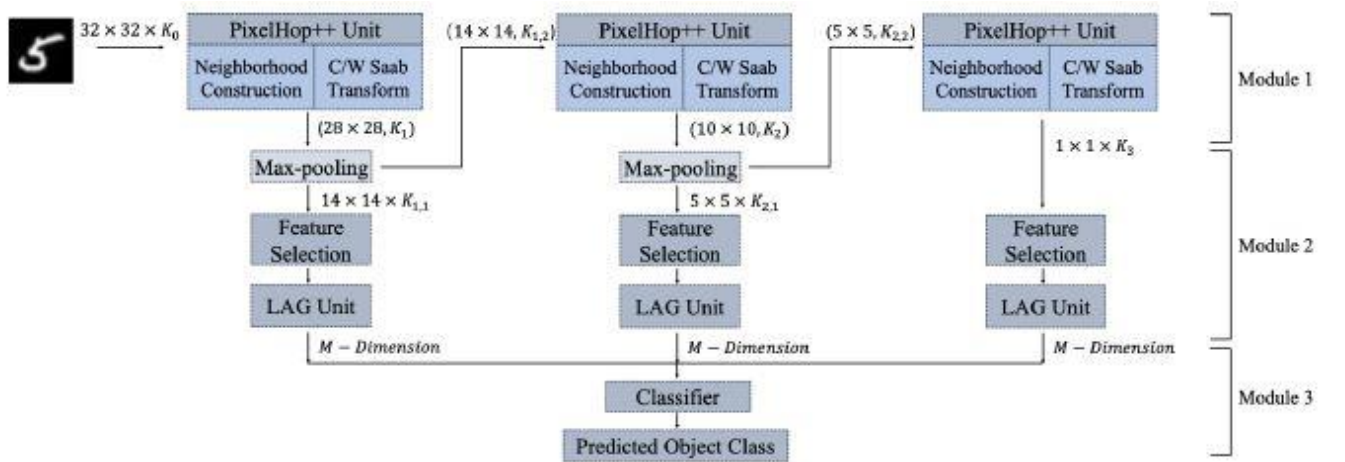


Figure 4. PixelHop++ Architecture

2.1.2 Approach and Procedures

Library: PixelHop++ contains several core modules, PixelHop unit, channel-wise Saab transform, LAG unit and machine learning techniques. For the first 3 parts, I use the opensource code of the github that published by the USC MCL. For the ML method, I use SVM instead of RF since there are less parameters to tune and more easily to manipulate.

Fitting Dataset: For the PixelHop model, we need to use a relatively small dataset to fit the model first. To construct a random selected dataset that contains 1000 images for all 10 classes, I use random library to generate a 1000-long sequence, which contains 1000 distinct numbers between 0 to 4999. For each class, these numbers of images are going to be selected and form a fitting dataset. This approach can make sure the fairness of each class and also assure the randomness of the dataset. Also, this function can be used to construct the dataset that contains of different portions of the original training set.

Neighborhood Construction: In this assignment, we use the sliding window with size of 5 and stride of 1 to construct the neighborhood representations. Also, the max-pooling operations are applied at the first two stages of the model, the max-pooling window size are both 2 by 2. What is necessary to mention is that the input of the next stage is the result tensor after the max-pooling operations.

Batch Method: In this part, to avoid the memory error while doing the channel-wise transformation. I implemented a batch method to divide the

total 50K images of the training set into 25 mega batches which contains 2000 images per mega batch. For each mega batch, the image data are divided further into 20 batches with 100 images per batch. For each batch, the c/w Saab transform is applied separately and the 50k raw representations of each stage are going to be merged together as one feature tensor. Which is going to be the inputs of the feature selection module.

Feature Selection: There are one part of PixelHop++ that wasn't discussed in Problem 1, which is the feature selection. In this Module, we calculate the cross entropy of each feature that extracted by Module 1. The features with low cross entropy usually means that they are most distinguishable between images and classes. So, we use training data to find the top 50% features with lowest cross entropy and record their indexes in the feature tensors. For test features, we select the same indexes of the feature tensors as the input of the LAG unit. This process can be regarded as another process for filtering the representative features and dimension reduction other than LAG units.

The detailed hyper-parameters of the model is shown as Table 1 below.

Table1. Detailed Hyper-Parameters of PixelHop Model

Spatial Neighborhood size in all PixelHop++ units	5x5
Stride	1
Max-pooling	(2x2) -to- (1x1)
Energy threshold for intermediate nodes ($TH1$)	0.001
Energy threshold for discarded nodes ($TH2$)	0.0001
Number of selected features (N_s) for each Hop	Top 50%
α in LAG units	10
Number of centroids per class in LAG units	5
Classifier	Random Forest (recommended)

2.1.3 Experimental Result

Note: For the feature selection, instead of using the top 50% of image features, I used the top 1000 features with the smallest cross entropy in each layer.

The performances of the PixelHop++ with different hyper-parameters are shown as Table 2. The training accuracy, testing accuracy, training time and inference time are all included. Since there are few differences between $N_s = 250$ and $N_s = 1000$, so, I calculated the average training time as results.

Table 2. The Performances of PixelHop++

Train Num	Feature Num	Train Acc	Test Acc	Train Time	Test Time
1560	250	97.82	49.50	23m 41s	54s
1560	1000	100	31.77		
3120	250	93.75	55.03	24m 58s	56s
3120	1000	99.42	49.70		
6250	250	87.81	58.90	28m	60s
6250	1000	96.16	57.76		
12500	250	82.92	63.02	34m	75s
12500	1000	90.83	62.59		
50000	250	78.52	67.90	82m 43s	82s
50000	1000	82.92	69.68		

The relation between the accuracies and the transformed images are shown as below in Figure 5. Since the model is easily to be overfitted with small amount of training samples, I ran two different parameters with 1000 and 250 selected features and plotted the relations between training sample numbers and training and testing accuracy.

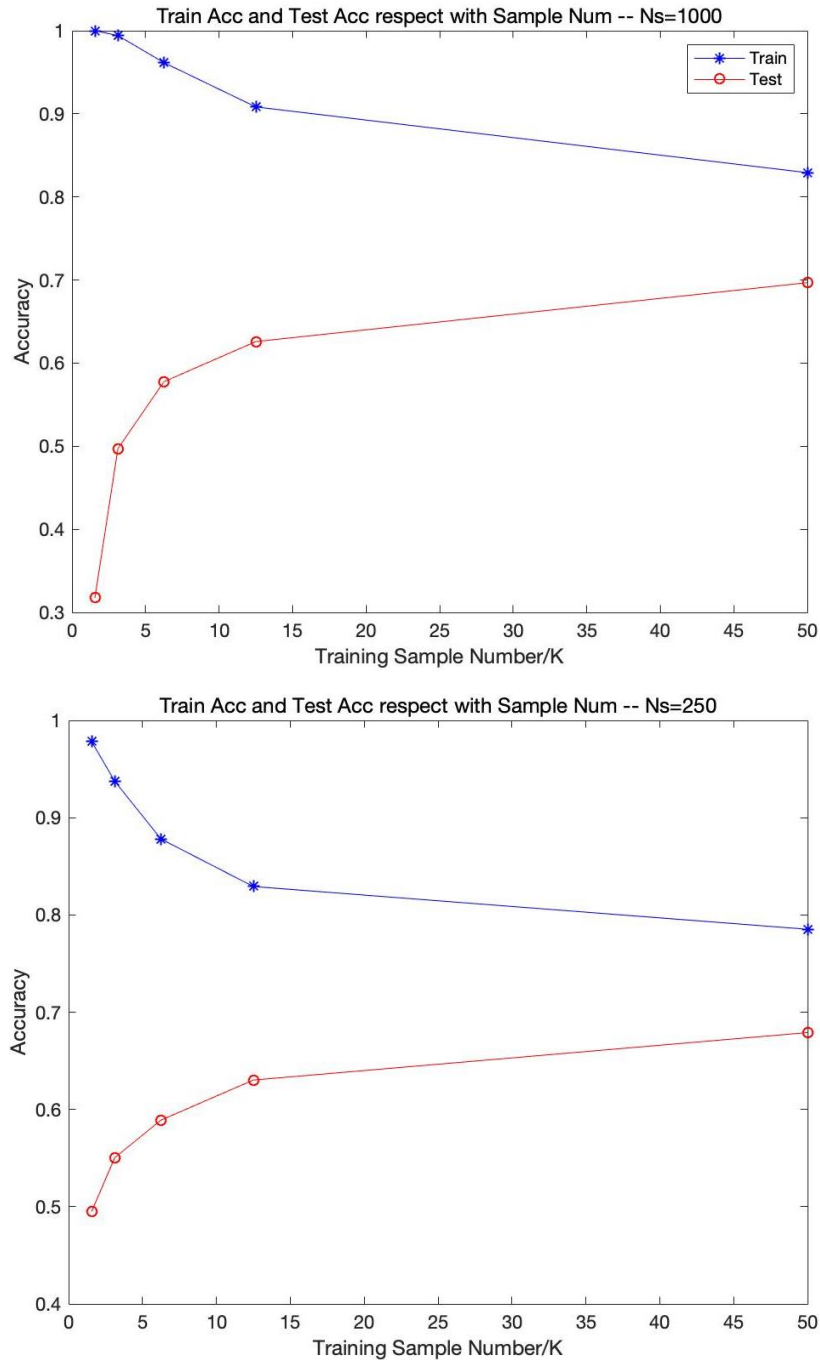


Figure 5. Train and Test Accuracy respect with Sample Numbers

For time consuming, the training time and testing time's relations between training sample numbers are shown as below in Figure 6 and Figure 7. Training time is measured by minutes and testing time is measured by seconds.

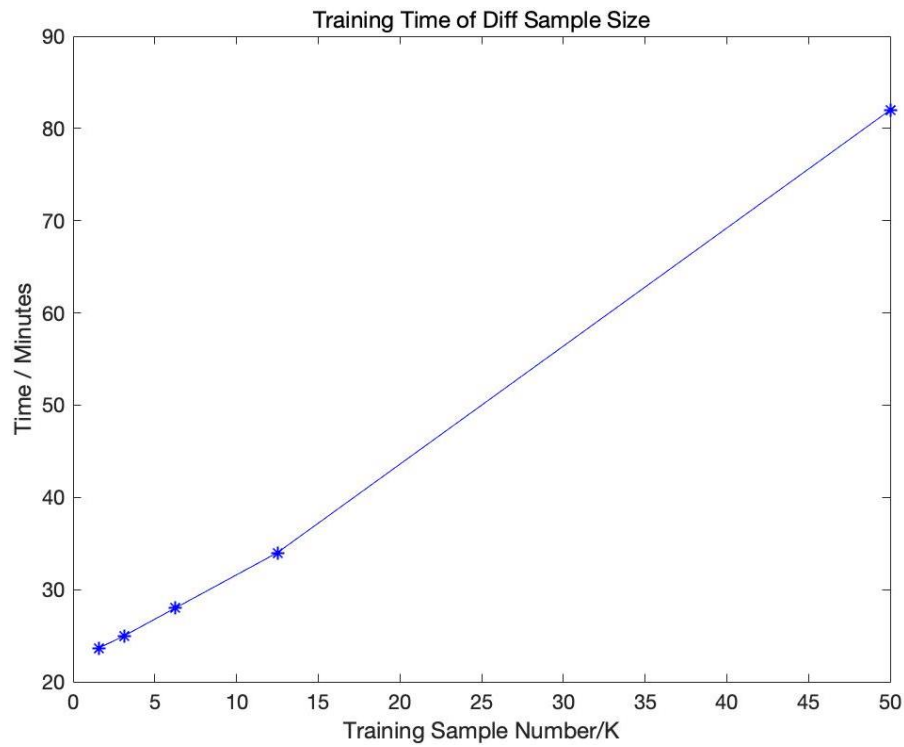


Figure 6. Training Time respect with Sample Numbers

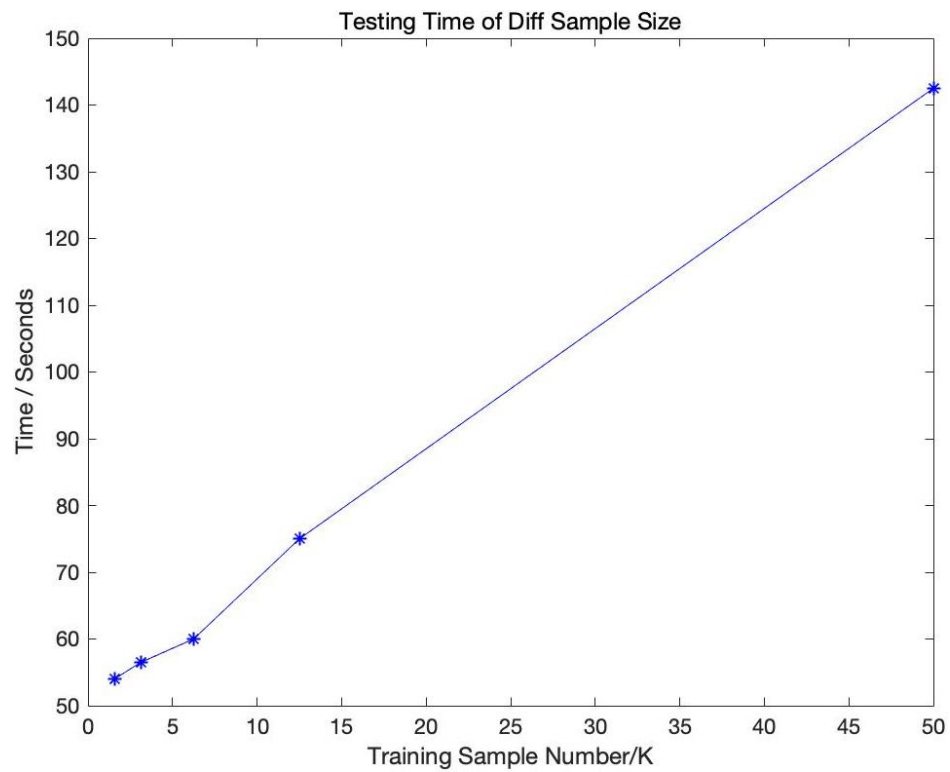


Figure 7. Testing Time respect with Sample Numbers

2.1.4 Discussion

Model Size Computation: Since for the first PixelHop++ unit, the Saab transform is not channel-wise, so the window size for PixelHop unit 1 is $5 * 5 * 3$. While the dimensions of output of each PixelHop unit is (50000, 28, 28, 42), (50000, 5, 5, 273) and (50000, 1, 1, 528). The later two PixelHop++ units use the channel-wise Saab transformations. So, the window size of the second and third PixelHop++ unit is $5 * 5 * 42$ and $5 * 5 * 273$, respectively. So the number of parameters for each PixelHop++ unit is computed as below in equation(1-3) respectively. So, the parameters of Module 1 is shown as below in equation(4).

$$N1 = 5 \times 5 \times 3 \times 42 = 3150 \quad (1)$$

$$N2 = 5 \times 5 \times 1 \times 273 = 6825 \quad (2)$$

$$N3 = 5 \times 5 \times 1 \times 528 = 13200 \quad (3)$$

$$N_{Module1} = N1 + N2 + N3 = 23175 \quad (4)$$

For each LAG unit, the output vector's size is $M=50$. So the total dimension of concatenated features are $3 * M = 150$. And the dimension of LAG inputs are set to 1000 using the cross entropy selection. Also, the total number of features in PixelHop++ Unit 3 is 528, which is less than 1000. Therefore, the number of parameters of Module 2 is computed as below in equation (5).

$$N_{Module2} = 2 \times 50 \times (1000 + 1) + 1 \times 50 \times (528 + 1) = 126550 \quad (5)$$

Since I used the SVM as the classifier, which has certain number of support vectors for each class. So, I used the `svc.dual_coef_` attributes to find

out the number of coefficients, which represent the number of parameters. The shape of `svc.dual_coef_` is (9, 30202). So the parameter number is $9 * 30202$, as shown in equation (6).

$$N_{Module3} = 9 \times 30203 = 271818 \quad (6)$$

So, the whole model's size is shown as Table 3.

Table 3. Model Sizes of Parts of PixelHop++

Part	Parameter Num	Module Params	Without Classifier	Total
PixelHop++ Unit 1	3150	23175	149725	421543
PixelHop++ Unit 2	6825			
PixelHop++ Unit 3	13200			
LAG Units	126550	126550	271818	
SVM	271818	271818		

Accuracy: From Table 2 and Figure5, we can see that the training accuracy is declining with the increase of the training samples. This is due to the overfitting problem is less severe while the training data is getting larger. Specifically, the training accuracy reaches 100% when $N_s = 1000$ and the training set is 1/32, and the testing accuracy is only around 30%. This means the SVM model overfitted the 1000 training features and lost most of its generalization ability. However, if the features dimension are reduced to 250, the testing accuracy is about 50% with the same number of training images. From the graph, we can still see that the overfitting problem exists since the

training accuracy is also very high. But from another prospective, we can say that the generalization ability of PixelHop++ is not bad since it can still maintain 50% accuracy from just 1/9 samples of the number of testing set.

From the Table 2, we can see that with small training set, the testing accuracy with $N_s=250$ is higher than the testing accuracy with $N_s=1000$. This is due to the overfitting is very easily to happen in small dataset, and reducing the dimensions of features can significantly solve this problem and increase the testing accuracy. However, if we reduce the dimensions of training images from 1000 to 250 with original training dataset, the testing accuracy will decrease. This is due to 250 dimensional features are insufficient for 50000 training images, and the SVM model turns to be under-fit the training data. This under-fitting problem can also be viewed from the training accuracy with $N_s=250$ is only 78%, lower training accuracy and lower testing accuracy means the under-fitting problem happens. From Figure 5, the testing accuracy is rising with the increase of the training sample number since the generalization ability is strengthened by more training images. So we can say that a large dataset is essential to PixelHop++ model.

Compared with the CNN, we can see that PixelHop++ model can still keep relatively high testing accuracy with very small training dataset. This means the robustness and the generalization ability is better than CNN. From the execution of the PixelHop++ code, I also found if we want to avoid the overfitting or underfitting problems, we only need to change the feature dimensions which is much more convenient than CNNs. Also, the testing

accuracies are slightly higher than LeNet-5.

Time: From Figure 6 and Figure 7, we can see that even with the whole training dataset, the training process is still within 90 minutes, which is very efficient compared with the CNNs running time on CPU. Also, we can say that the training time and the testing time is proportional to the training image number.

2.2 Error Analysis

2.2.1 Abstract and Motivation

Error analysis is very important in the image classification, since it can interpretate the model confusion by the human perspective. From this analysis, we can develop the more mature techniques to improve the model. For classification problem, the confusion matrix is a useful tool that can help us to view the classes with high accuracy or low error rate.

In this part, I use confusion matrix to illustrate the PixelHop++'s error model and find the classes with the highest and lowest accuracy. After that, some exemplary images of high error rate classes are going to be shown to explain why the errors are more easily to happen in these classes. At last, some ideas that might be helpful to improve the performances of PixelHop++ model are proposed and justified.

2.2.2 Approach and Procedures

Library: For the confusion matrix plotting, I use the open-source package of python, which is called ‘seaborn’.

Normalization: The confusion matrix is normalized between 1 and 0. Since the 10000 test images are constructed by 1000 images per class, so the we only need to divide every number in the original confusion matrix by 1000 as equation(1). What needs to mention is that the annotations in the confusion matrix is formatted as 2 numbers floats, so there might be some error in the approximation.

$$Norm_C_{ij} = C_{ij}/1000 \quad (1)$$

2.2.3 Experimental Result

The normalized confusion matrix is shown as below in Figure 8.

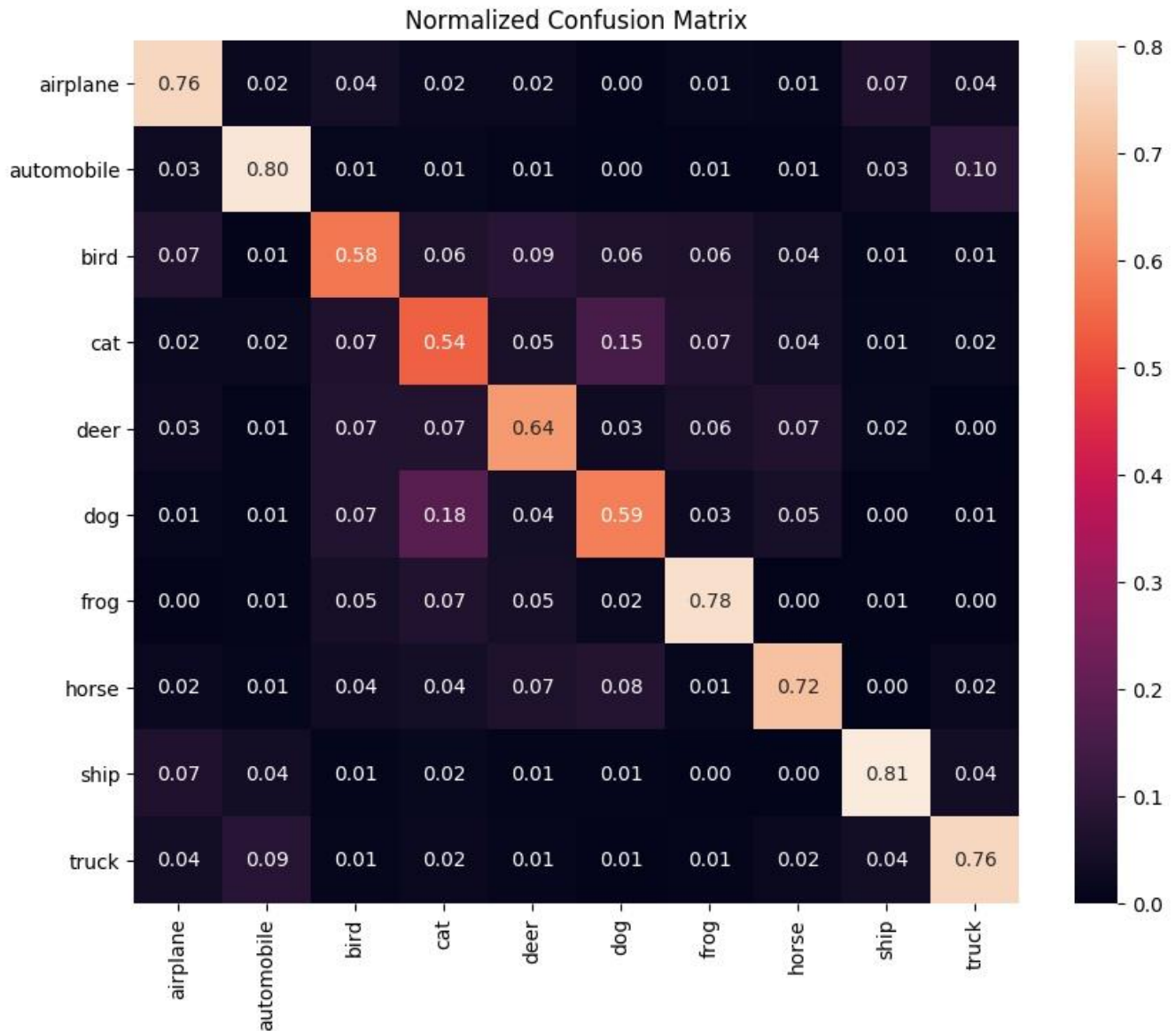


Figure 8. Normalized Confusion Matrix

2.2.4 Discussion

Classes with Highest/Lowest Error Rate: From Figure 8, we can see that the ‘**ship**’ class has the highest accuracy with 0.81, which means the error rate of this class is the lowest. However, the ‘**cat**’ class has the lowest

accuracy with 0.54, which means many samples of this class is mis-classified. In other words, ‘cat’ is the most difficult class. Also, we can find the ‘bird’ and ‘dog’ class are both difficult for the PixelHop++. But ‘bird’ is different from ‘cat’ and ‘dog’, which are easily mutually mis-classified. ‘Bird’ class is very easily to be mis-classified and the error doesn’t concentrate on one specific class, but can be mistakenly classified many other classes with similar probabaility, like with 0.06 error rates on ‘cat’, ‘dog’ and ‘frog’ and with 0.07 error rate on ‘airplane’.

Confusing Groups: I selected the four classes with lowest accuracy in the confusion matrix and found their easily confused classes and some classes share mutual errors to form the confusion groups. The confusion groups are shown as below in Table 4.

Tabel 4. Confusion Groups

True Class	Confusing Classes
Cat	Dog, Bird, Frog
Dog	Cat, Bird
Bird	Airplane, Cat, Deer, Dog, Frog
Deer	Bird, Cat, Frog, Horse
Ship	Airplane
Airplane	Ship
Truck	Automobile
Automobile	Truck

Some exemplary images of each confusion group are shown as below in Figure 9.





















































































































































































True	Exemplary Images of Confusion										Predict
Cat											Dog
Cat											Bird
Cat											Fog
Dog											Cat
Dog											Bird
Bird											Airplane
Bird											Cat
Bird											Deer
Bird											Dog
Bird											Frog
Deer											Bird
Deer											Cat
Deer											Frog
Deer											Horse
Ship											Airplane
Airplane											Ship
Truck											Automobile
Automobile											Truck

Figure 9. Exemplary Images of Confusion

From Figure 9, we can see that ‘cat’ and ‘dog’ class can be easily misclassified due to they share many facial similarities. Also, the bodies of cats and dogs are very similar, the bodies of exemplary images shown in first row of Figure 9 are very similar to dog images. And for the ‘bird’ images which

are mis-classified as ‘airplane’, most of them has skies or blue backgrounds. This might be same as the airplane data images. Of course, one of the main reasons of the errors is the low resolution of the dataset, human judgment can also easily be wrong on these exemplary images.

For other mis-classifications, the justifications and analysis are shown as Table 5.

Table 5. Justification of Mis-classifications

True Class	Predicted	Justification
Cat	Dog	Similar facial characteristics
Cat	Bird	Blue backgrounds or wing-like objects
Cat	Frog	Only cat faces, which might be like frog-like and one mis-labeled data image
Dog	Cat	Similar facial characteristics
Dog	Bird	Blue backgrounds or wing-like ears
Bird	Airplane	Sky or blue backgrounds
Bird	Cat	Brown backgrounds or furry textures
Bird	Deer	Struthio camelus or obvious bird legs which are the features of deer
Bird	Dog	Bird faces or furry textures
Bird	Frog	Green backgrounds or green feather birds which is the typical color of frog
Deer	Bird	Blue backgrounds or wing-like horns
Deer	Cat	Brown fur or deer faces which are likely to match to cat faces
Deer	Frog	Green backgrounds or head and body overlapping
Deer	Horse	Similar body shape and number of legs
Ship	Airplane	Ship pointy heads which are mostly the head of planes and sky backgrounds
Airplane	Ship	Sea or blue backgrounds which are similar to sky
Truck	Automobile	Truck heads and wheels which are similar to cars
Automobile	Truck	Similar body structure and the wheels are similar to trucks

Improvement Proposals: From the error analysis above, we can see that the generalization ability of PixelHop++ can still be improved by the data augmentation. Since some images with blue backgrounds are easily be judged as airplane and some images with green backgrounds are easily be classified as frog. These outliers can be recognized by the model if the generalization

ability of the model improves, which can be achieved by the data augmentation.

Another idea is that we can use two windows to represent the same image. As the inspiration of NLM algorithm, the images contain local information and global information. If the window size is fixed, like 5 by 5, the features can only be obtained in local areas and little information about the global features. As we can see above, the cat and dog images are difficult since they both have similar fur, eyes, noses and ears, while the face structure can be distinguished between them. But the small sliding windows cannot get enough information of global features like structures and backgrounds. To get these information, we need a larger sliding window in neighborhood construction process. So, to get both kinds of features, we can use a small window to extract local features like noses and ears, and a large window to extract global features like backgrounds and feathers. Of course, this procedure will cost more time in computation. However, we can use a bigger stride and with both window sizes to accelerate the neighborhood construction algorithm. Also, the procedure of feature selection and Saab transform can also be accelerated by using certain threshold and smaller N_s values. This proposal should work better and improve the generalization ability of the model since it views the data images in multiple scales and considers the global features into account.

References

- [1] C.-C. Jay Kuo and Yueru Chen, “On data-driven Saak transform,” *Journal of Visual Communication and Image Representation*, vol. 50, pp. 237–246, 2018.
- [2] 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.
- [3] 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.
- [4] Yueru Chen, Mozhdeh Rouhsedaghat, Suyu 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
- [5] Yueru Chen, Yijing Yang, Wei Wang, C.-C. Jay Kuo, “Ensembles of Feedforward-designed Convolutional Neural Networks”, in *International Conference on Image Processing*, 2019