Qidi Han (6814891757)
EE-569
Homework #6
April-22-2020

Problem 1: Understanding SSL

I. Abstract and Motivation

As the last homework, we learn LeNet-5 CNN. In this homework, we will learn the two type Successive Subspace Learning. Those two types of learning are new and powerful in image classifier right now. The image classifier and image recognition are important to our modern life. There are many different ways using the image recognition and image classifier. It has many major contributions in AI. The PixelHop and PixelHop++ are two major parts in the SSL. PixelHop++ will have a better result than PixelHop. We will read those two papers that wrote by Dr. Kuo. Then I will do some comparison for bot PixelHop and PixelHop++;

II. Approach and Procedures

a. FF-CNN

I will read one paper about Saab transform, and I will answer the two question, I will write the conclusion in the discussion part. Also, I will compare the FF-CNN and BP-CNN.

b. SSL

I will read two paper relate to the successive subspace learning, the first paper is about the PixelHop and the related information, the second paper is about the PixelHop++ that improved based on PixelHop. Then I will answer some question in the discussion part. I will list all the figures before the discussion.

III. Experimental Results

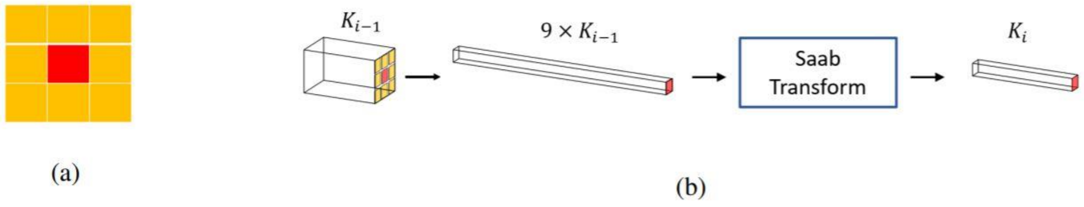
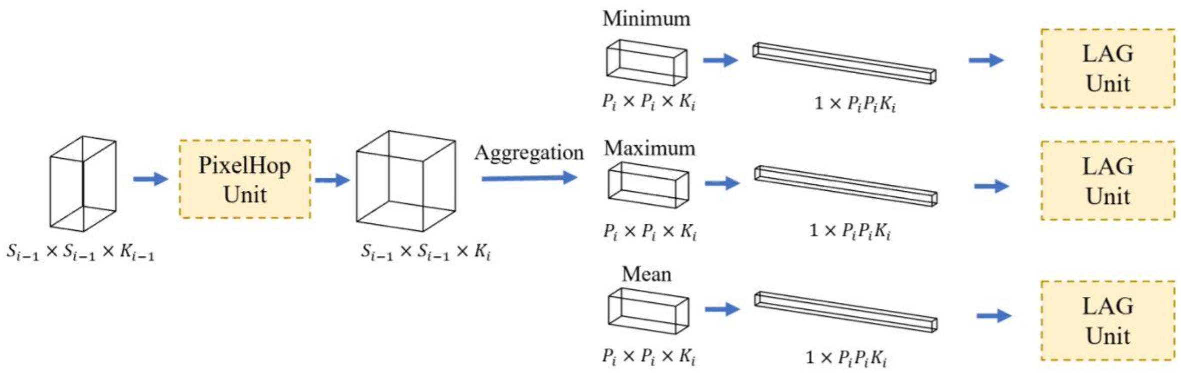
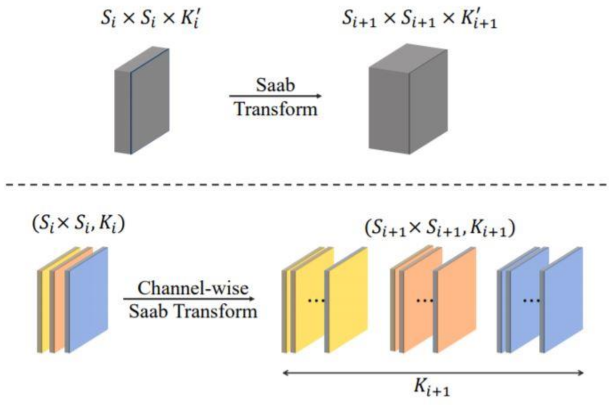


Figure 1 shows the block of one PixelHop



$S_{i-1} \times S_{i-1} \times K_{i-1}$
PixelHop Unit
$S_{i-1} \times S_{i-1} \times K_i$
Aggregation
Minimum
Maximum
Mean
$P_i \times P_i \times K_i$
$1 \times P_i P_i K_i$
LAG Unit

Figure 2a shows the aggregation in module 2



$S_i \times S_i \times K_i'$
Saab Transform
$S_{i+1} \times S_{i+1} \times K_{i+1}'$
$(S_i \times S_i, K_i)$
Channel-wise Saab Transform
$(S_{i+1} \times S_{i+1}, K_{i+1})$
K_{i+1}

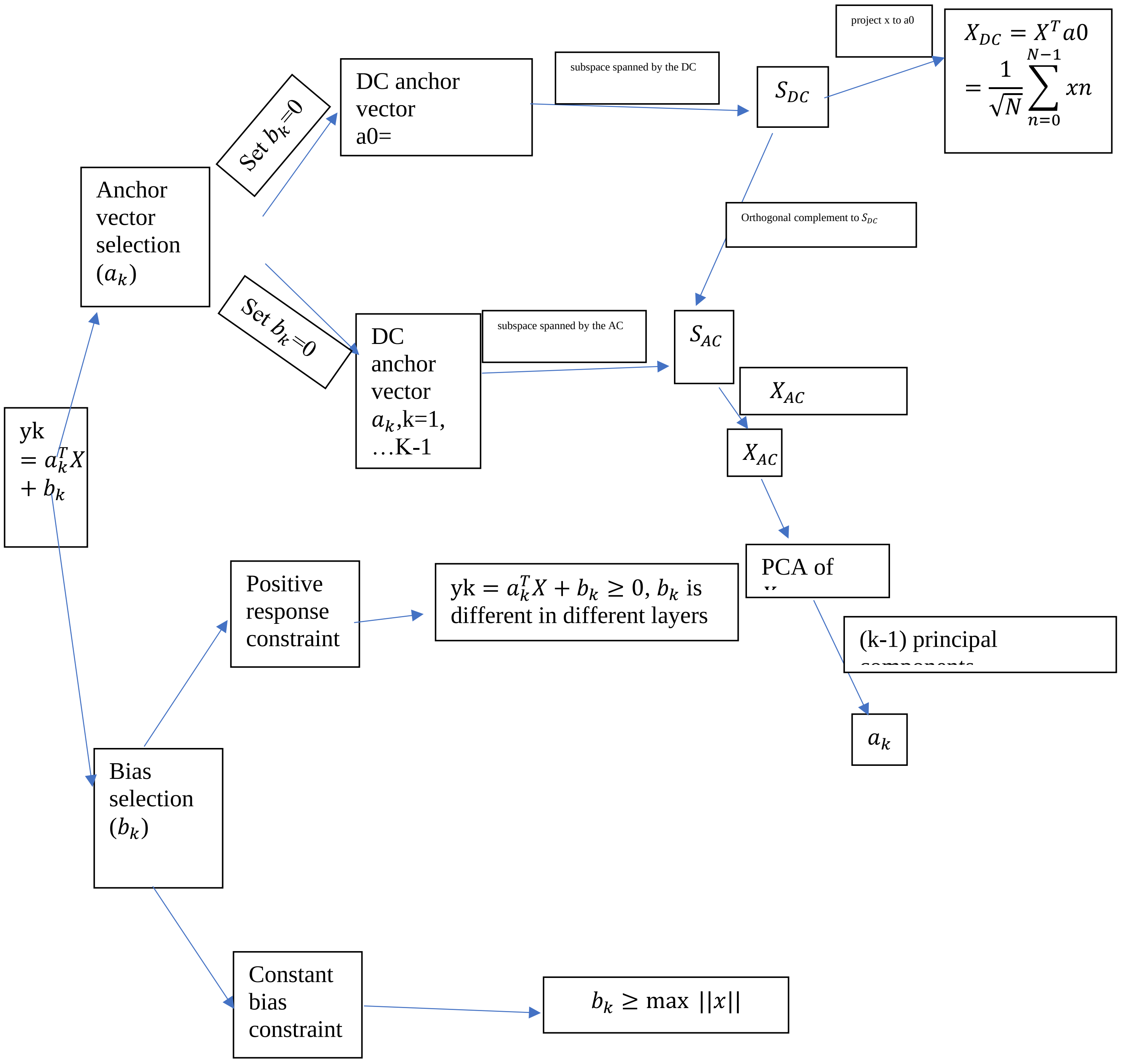
Figure 2b shows the comparison between traditional Saab and channel wised Saab transform.

IV. Discussion

a

(1)

For the saab transform, there are two parts. The first part is the selection of a_k. The second part is the selection of b_k. For the selection of a_k. We first set the bias term as 0. Then, there are two different anchor vectors, such as DC and AC anchor vector. We will separate it into those two terms. Next, we will find the subspace spanned for both anchor vectors. We will describe them as Sdc and Sac. For the data points in the DC subspace, we will project the data points x to DC subspace. And for the data points in the AC subspace, we will use the $X_{AC} = X - X_{DC}$, because Sac is orthogonal complement to Sdc. Finally, we will do pca for the X_{AC}, and select the first k-1 term of the pca. For the selection of bk, there are also two method. The first method is that setting the yk greater equal than to 0. This bias will change in every layer. The second method is set a constant bk. We will get the bk greater equal thank max of X. X represent data pointers.


project x to a0
$X_{DC} = X^T a0 = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} xn$
DC anchor vector a0=
subspace spanned by the DC
S_{DC}
Set b_k=0
Anchor vector selection (a_k)
Orthogonal complement to S_{DC}
Set b_k=0
DC anchor vector a_k,k=1, …K-1
subspace spanned by the AC
S_{AC}
X_{AC}
X_{AC}
yk $= a_k^T X + b_k$
PCA of
Positive response constraint
yk $= a_k^T X + b_k \geq 0$, b_k is different in different layers
(k-1) principal
a_k
Bias selection (b_k)
Constant bias constraint
$b_k \geq \max ||x||$

(2）
Similarities:

Structure: The FF-CNN and BP-CNN all have three layers. The input layer, hidden layer and output layer, but BP-CNN just goes opposite way.
The hidden layer: Both BP-CNN and FF-CNN are containing the hidden layer, they are all very hard to explain and see through from the hidden layer. But we can use cross-entropy value to compute the hidden layer.
Robustness: they both are low robustness, because the adversarial attacks will bring a really bad result.
Modularity: FF-CNN have two main modules, the first module is the feature extraction, the second module is the regression and classification, each module will not depend on the input data. Both modules are all important for FF-CNN, and BP-CNN have 2 modules, But the input data will affect the module performance a lot. The output layer will more depend on the weight vectors. The modules are not that important to our process. The most important thing of BP-CNN is the weight vector and input data.

Difference:
The way of learning algorithm: The BP-CNN focus on the structure learning or the system optimization. BP-CNN basically just need to find a good network and good function to compute a good result, but FF-CNN focus on the analysis of data. There are many PCA in the FF-CNN. FF-CNN use the data probability analysis as the most important feature.
Mathematical: The BP-CNN needs many calculations; the weight function and cost function are most important thing. The gradient descent may cause many difficult issues. But FF-CNN just use the linear algebra. As I mentioned in the last topic, FF-CNN focus on the data probability analysis.
Time consumption: The FF-CNN will much fast than BP-CNN. There are many weight vectors need to be computed, and there are many different information that contain in the weight vector. Since there are many weight vectors, we need more data to train the model.
Structure: I already mentioned that there are three layers for both CNN, but the BP CNN is end-to-end optimization, but for the FF-CNN, we can choose different classifier for the determination layers. For example, we can use the SVM or the random forest classifier.
Generalizability: FF-CNN is easy on this case, the output will only depend on the input data for FF-CNN, but BP-CNN will need to build new networks for each input data, it's not easy things.
Accuracy: The BP-CNN is state-of-the-art, the accuracy of BP-CNN will better than the FF-CNN. But BP-CNN need more time to compute, and need more resource to do the process.
Dimension reduction: As the article mentioned that the FF-CNN will have higher cross entropy for 2 FC layer. FF-CNN have more evenly distributed among all the dimension of each layers.
Higher accuracy with frequent class: BP-CNN will have higher accuracy for the same class image that appear multiple times. But FF-CNN will not affect the accuracy, because we do the feature selection first, then do the classifier. It's nothing to do with our original images.
b)
(1)
The SSL methodology is totally different design with CNN. In the CNN, we have many different layers, each layer is depending on the last layer. But SSL doesn't contain any this kind of process. Instead of generating different models for each step. There are four steps of SSL. The first step is expending the neighborhood pixel by many PixelHop method, it just expands the neighbors according to the target pixel. The second step is reducing the dimension by unsupervised. The third step is reducing the dimension by LAG. This LAG also is a supervised.

The final step is the similar with CNN, it the combination of the feature and present the probability of decision. I will fully explain the detail below.
The first step: There are many PixelHop in the first model. The first PixelHop contains S(i-1)*S(i-1)*k(i-1) values. Each PxielHop has some relationship, I will describe the relation in next step. The main idea of this step is extracting the features of the images. Each image will generate many PixelHop. For each PixelHop, there is a target node marked as 0-hop input, then we will use a line to connect each pixel with the shortest path by using consecutive edges n times. And we can set a size for the neighborhood region size for each PixelHop. Larger region size will cover more neighbor pixels. Each pixel contains k dimension. As the Figure 1 shows the block of one PixelHop, the region size is 3*3, so there are 9*k dimension from this PixelHop.
The second step: As I mentioned in the question 1, the saab transform can reduce the dimension of X_{AC} by using the pca. After the reduce the dimension, we will have Ki dimension. After the unsupervised saab transform step, there is a max-pooling step after it. It will reduce the dimension from S(i-1)*S(i-1) to Si*Si between each PixelHop. It will not use for aggregation step, but it will generate the next PixelHop. The reason of doing this max-pooling step is because each nearby pixel is overlapping to each other. Figure 1 shows this step.
The third step: Reducing the dimension by surprised learning by using LAG method. After the step 2, we have S(i-1)*S(i-1)*k(i) value for each PixelHop. We will take the minimum, maximum and mean of each k dimension. Each k dimension, we can set a Pi to get Pi*Pi minimum, maximum and mean of each k dimension. Then, we will have Pi*Pi*Ki pixels. We will make it in 1-dimension space (1*PiPiKi). Figure 2a shows the figure of this process.
The final step: For each PixelHop, we will have PiPiKi features. We will write PiPiKi to M. There are I PixelHop. The totally M*I feature. Next, we will use the SVM classifier with radial basis function. Finally, we will get the result from the SVM classifier.
Compare DL and SSL: There are 12 different aspect of the comparison. I will brief describe each aspect below.
Model complexity: For the SSL, we have to pre-defined many parameters., and it's easy to add and modify the units for each part. The parameters of DL are more than training samples. SSL can be described as Non-parametric mode. And DL can be described as parametric mode.
Difficulty of learning: DL is hard to train the data if there we insert new class and images. But SSL will easy train any new data.
Architecture of Model: SSL is much flexible. DL is fixed model, it hard to change the model of DL.
Hidden layer: there are some hidden layer and black-box, SSL doesn't have hidden layer and black-box.
Parameter search: DL use backpropagation process. SSL used feedforward process.
Number of Training and test data: DL need more resource to compute, because it use the backpropagation process, but SSL will use less data resource, because it use feedforward process.
Dimension reduction: DL use the matrix filter to compute the next layer by using the convolution, and SSL use projections of important components of subspace.
Task-independent features: DL will set the parameters by the input images and labels. To extract the feature, DL is task dependent. But SSL can do the task-independent and task-dependent.
Multi-tasking: it's difficult to do multi-tasking for DL, each cost function is determined by last step, but SSL can easy do multi-tasking.

Data attacking: The DL will change the parameter by small change of data, but DL will weak those attacking, because the PCA will get rid of those attacking.

(2)

I had mentioned some function for each module in last question. For module 1, there are many PixelHop units, each PixelHop have a max pooling between them. Usually, we use 2*2 to 1*1 max pooling. Each PixelHop contains S(i-1)*S(i-1)*k(i-1), the next PixelHop will contain S(i)*S(i)*k(i).

Module 2: There are two parts of module 2, the first part is the aggregation, the second part is the LAG unit. Figure 2a shows the process. I already describe the aggregation and LAG unit in last question, it's basically reduced and pick the most important features.

Module 3: it takes all the dimension and features from the LAGs and put it into the classifiers. We can choose to use SVM or random forest, etc.

(3)

PixelHop: I already fully explain the neighborhood construction and subspace approximation step in last question. The PixelHop connect each pixel with the shortest path to do the neighborhood construction, it also calls near-to-far neighborhood expansion. It also uses the Saab to reduce the dimension. As I mentioned in the first step, the saab reduce the dimension of X_{AC} by using the pca. As the author mentioned that the spectral domain in saab transform has weakly correlation, but strong in spatial domain.

PixelHop++: The PixelHop++ have similar way to do the neighborhood construction, but it will include more channels, because we will increase the number of channels for the c/w saab transform. The subspace approximation will also increase the number of channels, it also increases the dimension of the transform. Figure 2b shows the comparison.

Comparison: There are more channel for PixelHop++ compare with PixelHop. The main different is that the input tensor size is different. The saab transform multiple the spectral and spatial dimension, but as I mentioned before, Spectral is weakly correlation in our transform, therefore, PixelHop++ will only use the spatial neighborhood pixels. It will enhance our result for feature extraction. From the dimension perspective, the PixelHop has S(i-1)*S(i-1)*k(i) term, but it's in one channel, but PixelHop++ has S(i-1)*S(i-1) for each channel. It used only spatial dimension.

(4)

I already explain the LAG for PixelHop in step 3 in question 2a. There are two important roles for LAG. The first role is making the local attributes to global attributes, since SSL represent successive neighborhood and subspace expansion. The second role is to reduce the dimension for concatenated attributes. We need to reduce the dimension to classifier each subspace and classes. The procedure of LAG is take the PixelHop unit with dimension S(i-1)*S(i-1)*k(i), and take the minmum, maximum and mean of those data to build Pi*Pi*Ki dimension, and convert is to 1*PiPiKi dimensions. The PiPiKi will describe as M to our module 3.

The advantage: it only did one integration and we can get all attributes for all the feature vector. It will enlarge the relationship between the feature of the images with different classes, because when we use dimension reduction, it extracts the most important information. Also, LAG helps us to find the data labels in SSL as the dimension reduced.

Problem 2: CIFAR-10 Classification using SSL

I. Abstract and Motivation

PixelHop++ model is famous in the modern day. As I mentioned before, those image recognition and image classifier are important feature of our modern learning. As the CIFAR 10 data set, there are a high accuracy for traditional training. Also, the tranditional method have many disadvantages. PixelHop++ will have many different choices for the parameters. The traditional method also had a hidden layer that the PixelHop++ doesn't exist. The user can see any data from each layer in PixelHop++. As I mentioned before, there are many advantages for PixelHop++.

II. Approach and Procedures

a. Build Model

In the first part of the experiment, I will use the GitHub opensource to build a PixelHop++. There are three modules in PixelHop++. The first module include channel-wise Saab transform and neighborhood construction. For this module, there are three different hops. Each hop will have a step of Max-pooling. The second module is the feature selection. There are two important things included in this module, the first thing is the k-means cross entropy. The second thing is the LAG unit. Before the module 2, there is a max-pooling step. The last module is the classifier module. The user can choose their own classifier, but in our experiment, I choose random forest to do the classifier. Figure 3 shows the process of the PixelHop++. I will explain the dimension from each module in discussion part. Figure 4 shows parameters for the PixelHop++.

1.

For the first question, I will build a PixelHop++ by using the pytorch in python. I will report the training time, train accuracy and model size for the PixelHop++. The process of this question will build the Saab transform first; each channel-wise Saab transform will build on the top of the Saab transform. Then, we will build PixelHop unit on top of c/w Saab transform. In the second and third PixelHop will use max-pooling method before the Saab transform.

After we build the PixelHop++ unit, we will do a max pooling. This max-pooling process is the same as the last step, but the last step is using the max-pooling for the next layers. This max pooling is reducing the dimension for feature selection. After we have the matrix for the feature selection, we will do the process called k-means cross entropy. K-means cross entropy use the mean of each feature cluster to classifier the feature. Then we will find the more important feature by finding the minimal number of all the cross entropy. We will reduce top 50% dimension from the feature selection. Next, we will send those feature dimension to LAG unit, to reduce more dimension. I already fully explain the LAG unit in last question. The output for each hop is M dimension. There are three different hops. Therefore, we have total 3*M dimension for our final step. I use the random forest for our last step. There are total 150 dimension for each picture, I set the number of estimators to 70. I will show the result in the discussion part. For the training time, I will set the clock to report the time. As the Ta mentioned that there are three part of our module will consider in our model size of the parameter. The first part is the dimension from three PixelHop++. The second part is the dimension from the three LAG unit. The third part is the dimension from the random forest classifier. The formula for the model size shows below. M represent the output dimension from LAG unit. N represent the dimension for features. Figure 5 shows the result for training data. Figure 4a shows the dimension before the max-pooling and after max pooling.

$$model_{size} = dimensionofkenel * dimension_{hop} + 3 * M(n+1) + 3 * 150 * 10$$

2.

After I have the train module for the process, I will compute the accuracy my test data. I will repeat the same step as the training data. There are also 3 different modules. But this time, we will not train the data, we just need to apply the filter or the module for our test set. The easy way to say is that we just need to do the transform function for each module. For the second module, we will use the same index that from the training data. For the random forest, I just use the rf.score function to compute the accuracy. Figure 6 shows the result for the test data.

3.

For this experiment, I will set the module 1 unchanged. But I will use 1/4, 1/8, 1/16, 1/32 of the train data to train the module 2 and 3. This is the same process as the question 1. The only different is the size of the images. For each task, I will compute the accuracy for the test data. Finally, I will plot each test accuracy vs the size of train images. I will discuss the result in the discussion part. Figure 7 shows the plot.

b. Error Analysis

1.

The first step of this question is use the rf.predict function to predict each images from our test data. Then, I will use those predict image with our truly test data to build the confusion matrix, I will use plot_confusion_matrix function to plot this. Also, I will use heat map to plot this matrix again, the heat matrix can clearly see the result of easy and difficult. I will normalize the confusion matrix. So, from the diagonal of the matrix, we can find the lowest error rate. From the matrix, the combination of error rate and the probability of other class, we will find the most difficult class. Figure 8 shows the normalized confusion matrix. Figure 9 shows the heat map for my matrix.

2.

In This problem, we will find the confusing groups from the normalized confusion matrix. And I will show an example with those confusing groups. Figure 10 shows the example of confusing groups.

3.

To improve the accuracy of the difficult class, there are many different ways. I will discuss some ways in the discussion part.

Experimental Results

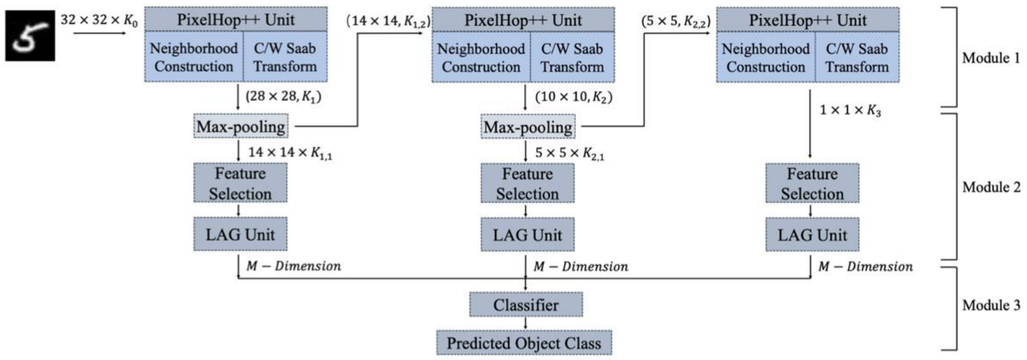
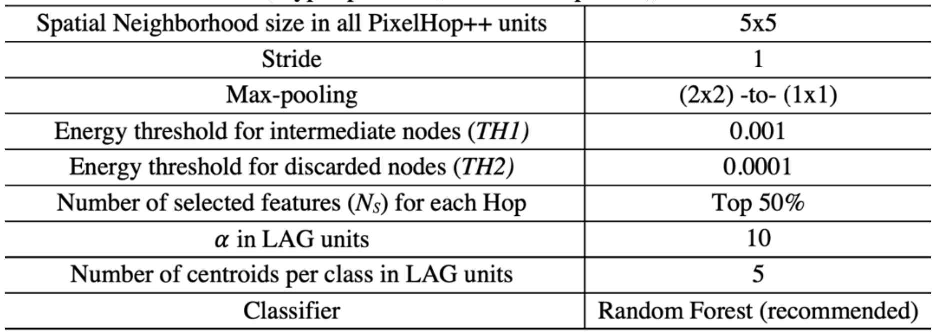
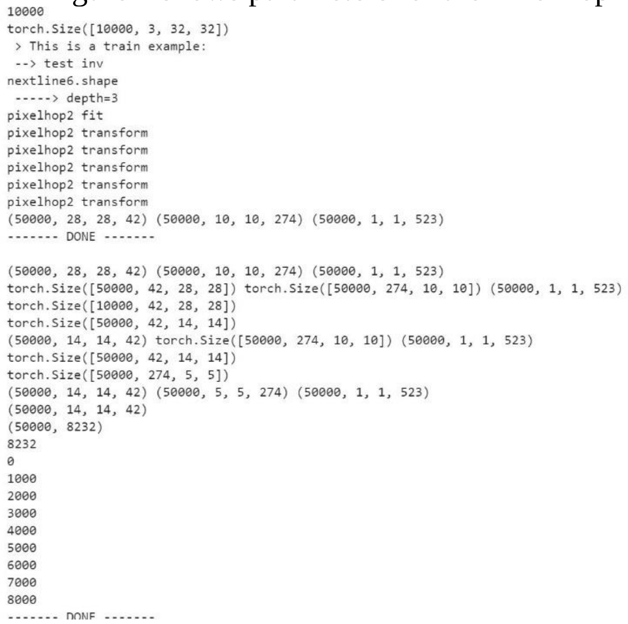


Figure 3 the step of the PixelHop++



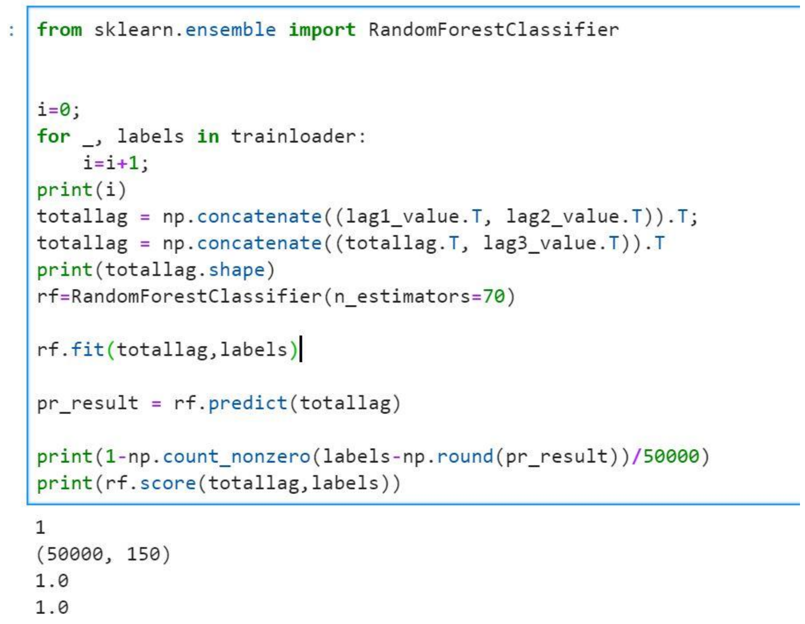
Spatial Neighborhood size in all PixelHop++ units	5x5
Stride	1
Max-pooling	(2x2) -to- (1x1)
Energy threshold for intermediate nodes (*TH1)*	0.001
Energy threshold for discarded nodes (*TH2)*	0.0001
Number of selected features (N_S) for each Hop	Top 50%
α in LAG units	10
Number of centroids per class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

Figure 4 shows parameters for the PixelHop++.



```
10000
torch.Size([10000, 3, 32, 32])
 > This is a train example:
 --> test inv
nextline6.shape
 -----> depth=3
pixelhop2 fit
pixelhop2 transform
pixelhop2 transform
pixelhop2 transform
pixelhop2 transform
pixelhop2 transform
(50000, 28, 28, 42) (50000, 10, 10, 274) (50000, 1, 1, 523)
------- DONE -------

(50000, 28, 28, 42) (50000, 10, 10, 274) (50000, 1, 1, 523)
torch.Size([50000, 42, 28, 28]) torch.Size([50000, 274, 10, 10]) (50000, 1, 1, 523)
torch.Size([10000, 42, 28, 28])
torch.Size([50000, 42, 14, 14])
(50000, 14, 14, 42) torch.Size([50000, 274, 10, 10]) (50000, 1, 1, 523)
torch.Size([50000, 42, 14, 14])
torch.Size([50000, 274, 5, 5])
(50000, 14, 14, 42) (50000, 5, 5, 274) (50000, 1, 1, 523)
(50000, 14, 14, 42)
(50000, 8232)
8232
0
1000
2000
3000
4000
5000
6000
7000
8000
------- DONE -------
```

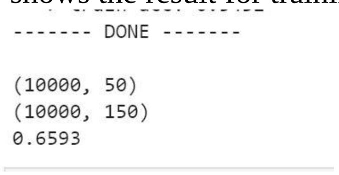
Figure 4a shows the dimension before the max-pooling and after max pooling.



```
from sklearn.ensemble import RandomForestClassifier


i=0;
for _, labels in trainloader:
    i=i+1;
print(i)
totallag = np.concatenate((lag1_value.T, lag2_value.T)).T;
totallag = np.concatenate((totallag.T, lag3_value.T)).T
print(totallag.shape)
rf=RandomForestClassifier(n_estimators=70)

rf.fit(totallag,labels)

pr_result = rf.predict(totallag)

print(1-np.count_nonzero(labels-np.round(pr_result))/50000)
print(rf.score(totallag,labels))
```

```
1
(50000, 150)
1.0
1.0
```

Figure 5 shows the result for training data.



```
------- DONE -------

(10000, 50)
(10000, 150)
0.6593
```

Figure 6 shows the result for the test data

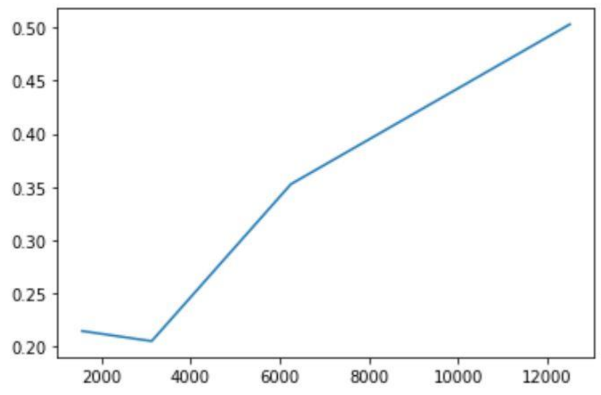
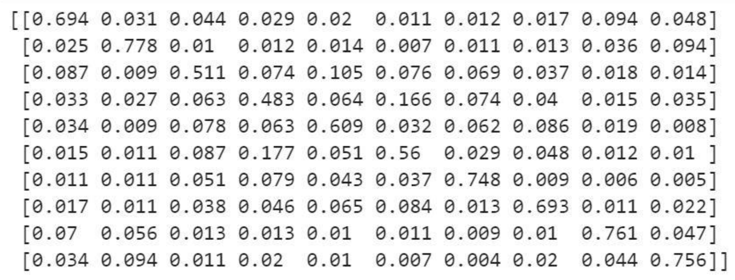


Figure 7 shows the test set accuracy vs size of training data.



```
[[0.694 0.031 0.044 0.029 0.02  0.011 0.012 0.017 0.094 0.048]
 [0.025 0.778 0.01  0.012 0.014 0.007 0.011 0.013 0.036 0.094]
 [0.087 0.009 0.511 0.074 0.105 0.076 0.069 0.037 0.018 0.014]
 [0.033 0.027 0.063 0.483 0.064 0.166 0.074 0.04  0.015 0.035]
 [0.034 0.009 0.078 0.063 0.609 0.032 0.062 0.086 0.019 0.008]
 [0.015 0.011 0.087 0.177 0.051 0.56  0.029 0.048 0.012 0.01 ]
 [0.011 0.011 0.051 0.079 0.043 0.037 0.748 0.009 0.006 0.005]
 [0.017 0.011 0.038 0.046 0.065 0.084 0.013 0.693 0.011 0.022]
 [0.07  0.056 0.013 0.013 0.01  0.011 0.009 0.01  0.761 0.047]
 [0.034 0.094 0.011 0.02  0.01  0.007 0.004 0.02  0.044 0.756]]
```

Figure 8 shows the normalized confusion matrix.

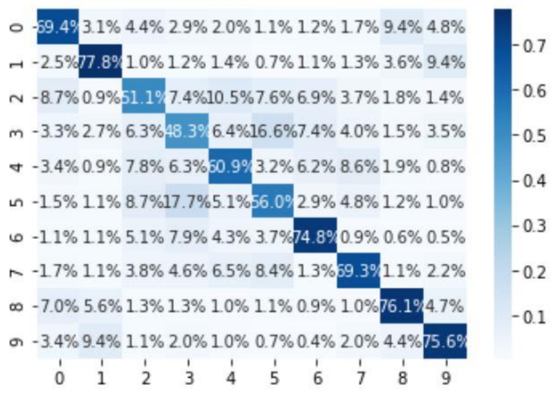
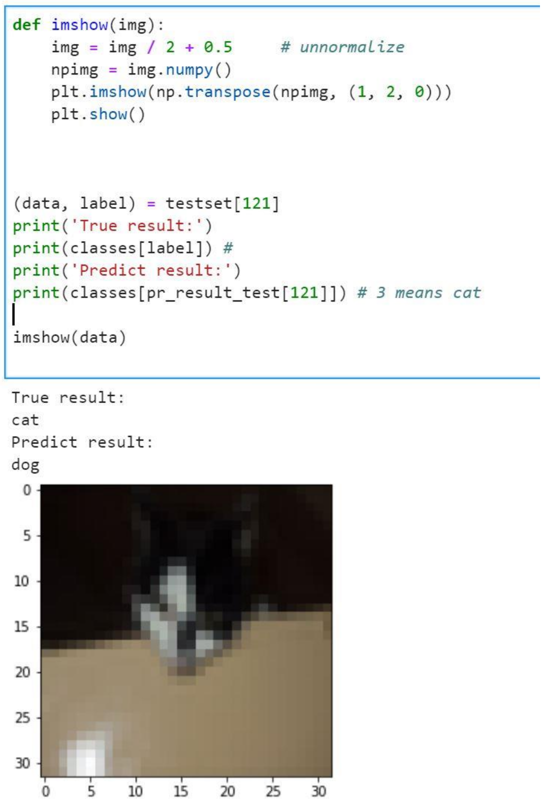
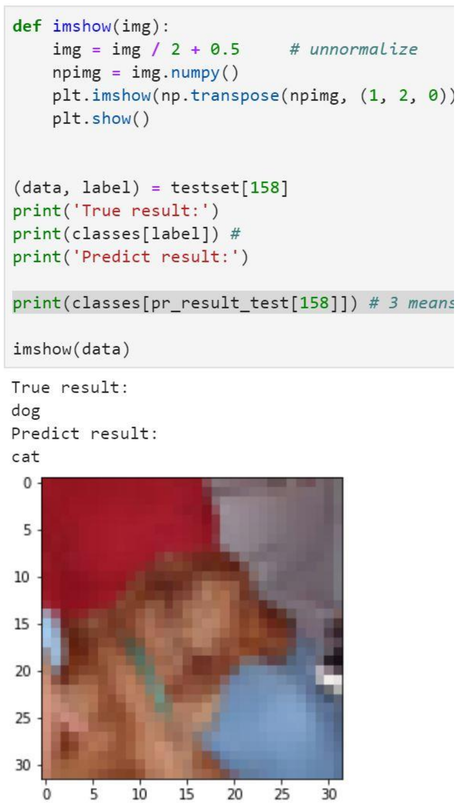


Figure 9 shows the heat map for my matrix.



```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()



(data, label) = testset[121]
print('True result:')
print(classes[label]) #
print('Predict result:')
print(classes[pr_result_test[121]]) # 3 means cat

imshow(data)
```

```
True result:
cat
Predict result:
dog
```



```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0))
    plt.show()


(data, label) = testset[158]
print('True result:')
print(classes[label]) #
print('Predict result:')

print(classes[pr_result_test[158]]) # 3 means

imshow(data)
```

```
True result:
dog
Predict result:
cat
```

Figure 10 shows the example of confusing groups.

III. Discussion

(a)

1.

I already fully explain the process in the approach section.

The training time is around 1hour 37 mins. I used 10k images from the train set to do the train.

Figure 5 shows the train accuracy and the code. I have 100% accuracy for train data. I used 70

number of estimators for random forest classifier. The model size is 270375. The detail of the model size is below:
Module 1: 5*5*3*42+5*5*274+5*5*523=23075
Module 2: 50*(4116+1+274+1+523+1) =245800
Module 3: 3*50*10=1500
2.
The test set accuracy is 65.93%. Figure 6 shows the result.
3.
Figure 7 shows the curve of test accuracy vs the training image. From the figure, we can tell that when the training image equal to ¼ of the 50k, the accuracy still has around 50%. There is no huge different with 50k training images, but when I use 1/8 of the 50k images, the accuracy only has 35%. Therefore, I may say that the 10k images still give us a good result, but 6k image will give us a worth result. When the image reduces to 3k and 1k, the accuracy reduces to around 20%, I guess, when the image reduces from 5k to 1k, the accuracy will close to 20%. The conclusion will be, when the training images are more than 13k, the accuracy for test data will greater than 50%. When the training image are less than 5k, the accuracy of test data will close to 20%.

(b)
1.
Figure 8 and Figure 9 shows the confusion matrix and heat map for our experiment. The lowest error rate is class 2 and class 8, which is automobile and ship. The accuracy of automobile and ship is 77.8% and 76.1%. The most difficult class is class 4, which is cat class. The accuracy for class 4 is only 48.3%. The reason of the high accuracy for automobile and ship is because the car has many special features. It's easy to separate with other images in dataset. But the cat images are hard to separate the features, from the image view, the dog and cat will have similar features in the data set.
2.
There is one confusing class group: The class 3 with class 5 (cat and dog class). From the Figure 9 shows that class 3 have 16.6% misclassified rate to class 5, and class 5 have 17.7% misclassified rate. Figure 10 shows the cat image is classified to dog class. From human eye, it's also hard to tell if this is a cat or dog. The features of cat and dog are very similar to each other's.
3.
There are is a major problem in our problem, the problem is cannot separate the dog and cat image. There is some solution that I want to mention, the first solution is we can add more layers to the structure. Before the classifier, we can add a layer to separate the animal class and transportation class. After we separate the animal class, we can separate to more small class, it's easier to classifier the image with only few classes if we already separate the image preview. For each class of the image, I will do the normalization to the image, because some image is very dark, some image is very bright. Those normalization will help us to classifier the images.
The second solution is we will repeat the same process as last solution, we still want to classifier the animal and transportation class. And for the animal class, I will add one more layer, it's for detecting the face area among all image, the output will give a decimal number represent the area of the face dived by the area of the image. From my perspective, I think the cat face will have small size among the image, the dog will have a bigger size among the image. But this is just one feature determination, we cannot determine the image by only use this method. For the

transportation image, this method also will be worked. the image size among all image for airplane image will smaller than the ship image. We can predefine the image each class before we use it.