EE 569 Discussion 12

Yijing Yang

04/10/2020

Contents

- Announcements
- Homework 6: Successive Subspace Learning
  - Problem 1&2
  - Will focus more on implementation details for P2

Announcements

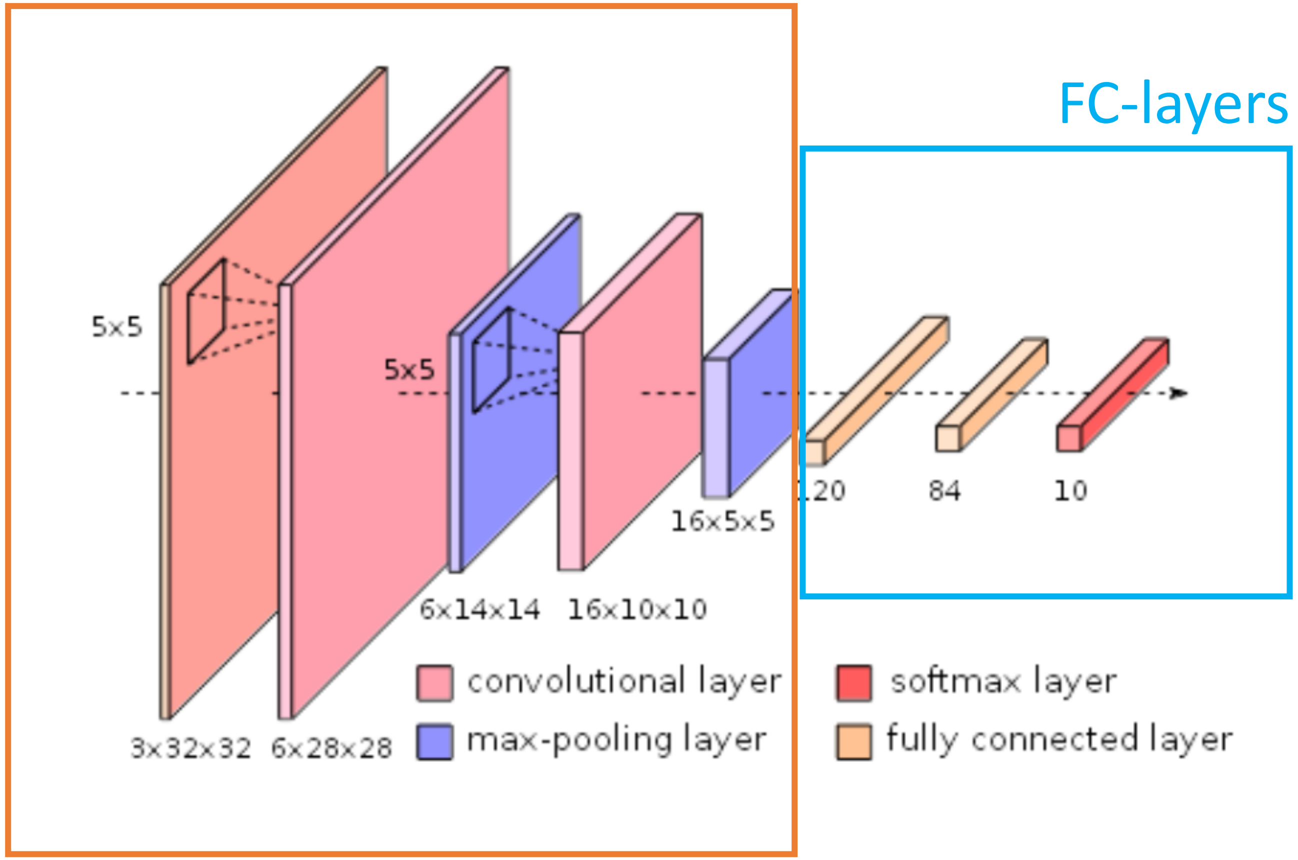
- HW6 P1 & P2 (100%)
  Issued: 04/08/2020
  Due: 04/26/20 11:59pm

- HW5 & HW6 Competition (50%+50%)
  Both due: 05/03/20 11:59pm

HW6 P1(a): FF-CNN and Saab Transform

Feedforward-designed Convolutional Neural Networks (FF-CNNs) [2]

FC-layers → Multi-stage LLSR



Conv layers →Multi-stage **Saab transform**

HW6 P1(a): FF-CNN and Saab Transform

Code for Saab Transform:

https://github.com/USC-MCL/EE569_2020Spring

- README.md
- cross_entropy.py
- cwSaab.py
- lag.py
- llsr.py
- pixelhop2.py
- saab.py

HW6 P1(b): SSL Methods

1. PixelHop [3]

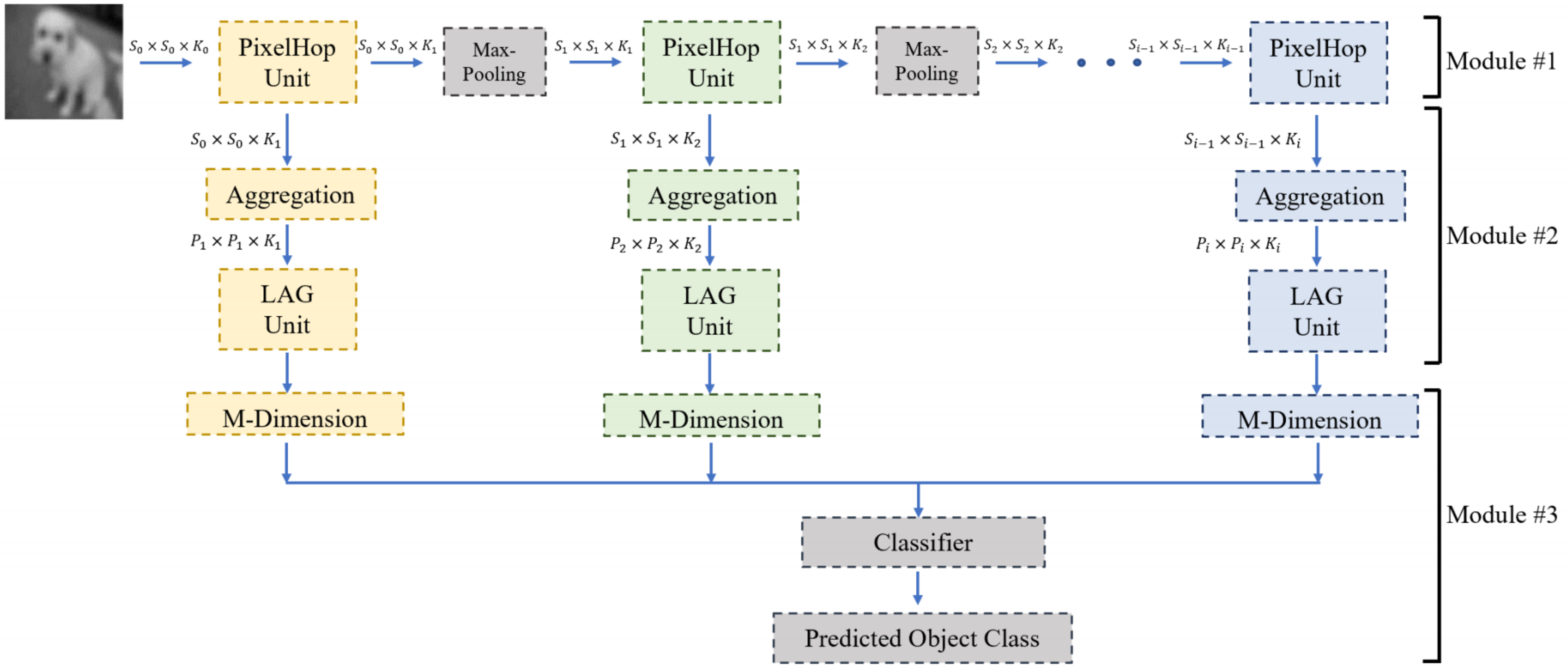


Figure 1: The block diagram of the PixelHop method.

HW6 P1(b): SSL Methods

2. PixelHop++ [4]

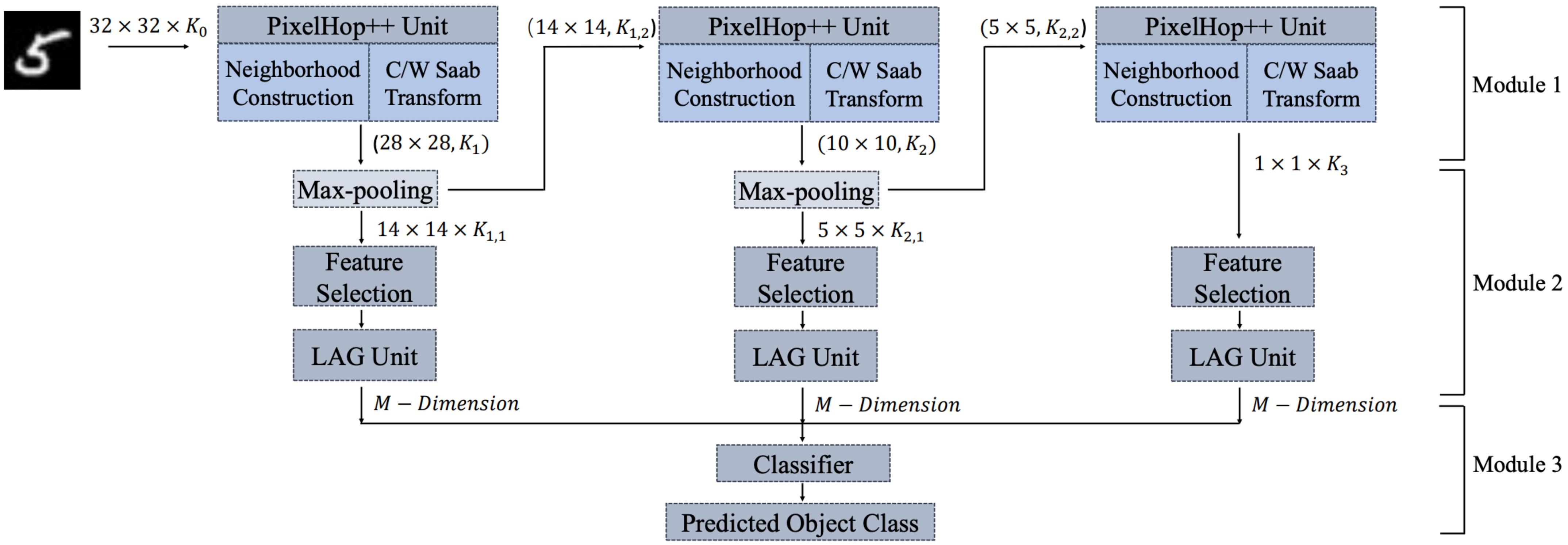


Fig. 1. The block diagram of the PixelHop++ method that contains three PixelHop++ Units in cascade.

HW6 P2: CIFAR-10 Classification using SSL

Things to be covered later:

- General guidance
- Explain the source code from high level
- Explain how to use the source code
- Implementation steps

HW6 P2: CIFAR-10 Classification using SSL

General guidance

- Source code for key modules are provided in the GitHub link
- The link is not the official implementation for the PixelHop++ paper
- Feel free to import them or modify based on them
- Your data should be channel-last,

  i.e. shape = *(N_image, Height, Width,* ***N_channel****)*

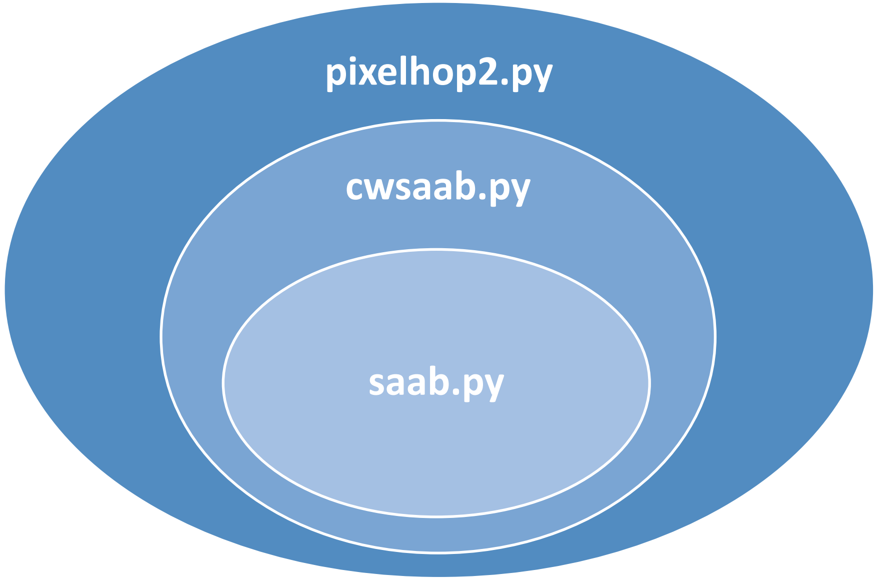
  instead of

  (N_image, ***N_channel,*** *Height, Width)*

HW6 P2: CIFAR-10 Classification using SSL

More about the source code

- cross_entropy.py — Feature-Selection Module
- cwSaab.py
- lag.py — LAG unit
- llsr.py — (tool) self-packaged LLSR
- pixelhop2.py
- saab.py



HW6 P2: CIFAR-10 Classification using SSL

More about the source code

Explanation of the code (watch the Discussion video for more details)

HW6 P2: CIFAR-10 Classification using SSL

More about the source code

About *Neighborhood Construction* (write your own ***Shrink*** function):

- Collect 5x5 (spatial) patches
- Pooling

Example:

```
from skimage.util import view_as_windows
# example callback function for collecting patches
def Shrink(X, shrinkArg):
    win = shrinkArg['win']
    stride = shrinkArg['stride']
    ch = X.shape[-1]
    X = view_as_windows(X, (1,win,win,ch), (1,stride,stride,ch))
    return X.reshape(X.shape[0], X.shape[1], X.shape[2], -1)

shrinkArgs = [{'func':Shrink, 'win':2, 'stride': 1},
              {'func': Shrink, 'win':2, 'stride': 1}]
```

HW6 P2: CIFAR-10 Classification using SSL

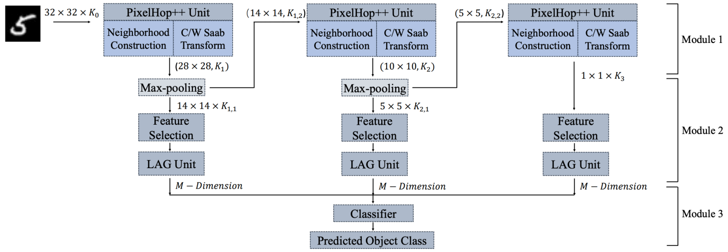
More about the source code

An example about how to import these functions:

```
from cross_entropy import Cross_Entropy
from lag import LAG
from llsr import LLSR as myLLSR
from pixelhop2 import Pixelhop2
```

```
if train:
    lag = LAG(encode='distance', num_clusters=[5,5,5,5,5,5,5,5,5,5],
              alpha=10, learner=myLLSR(onehot=False))
    lag.fit(selected_feature, train_labels)
```

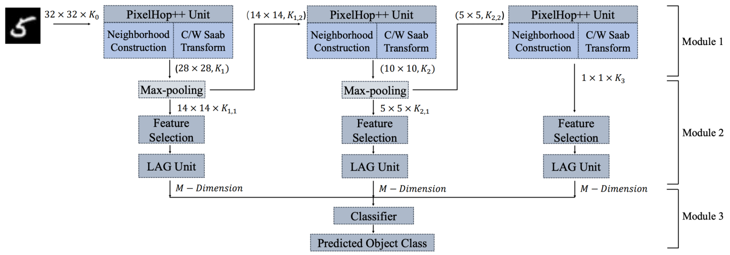
HW6 P2: CIFAR-10 Classification using SSL



★ Implementation Steps

1. Train Module 1:
   i. Define your Neighborhood Construction function, *SaabArgs, ShrinkArgs, ConcatArg*
   ii. (Pre-process data): channel-last, int→float, rescale to 0-1, etc.
   iii. Randomly select 10K training images: need to be balanced (1000 per class)
   iv. Use *pixelhop2.py* to train (*pixelhop2.fit*) your PixelHop model of a certain depth
   v. Save the PixelHop model

HW6 P2: CIFAR-10 Classification using SSL

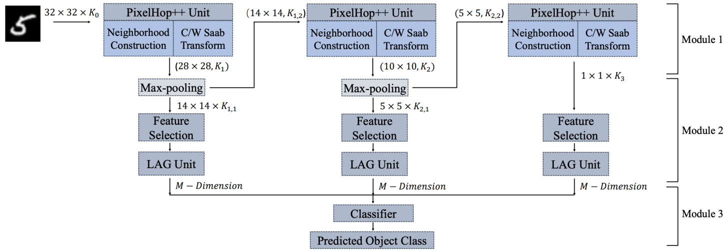


★ Implementation Steps

2. Extract features from Module 1:

- Use all the 50Ktraining images, do *pixelhop2.transform* to extract features
- Outputs are before max-pooling
- May need batch processing to save memory. Try different programming methods on your own

HW6 P2: CIFAR-10 Classification using SSL



★ Implementation Steps

3. Module 2, for each Hop:

i. Do feature selection using *cross_entropy.py:*

-- Use *KMeans_Cross_Entropy* to calculate the cross-entropy for each feature dimension;

-- Sort the cross-entropy in ascending or descending order;

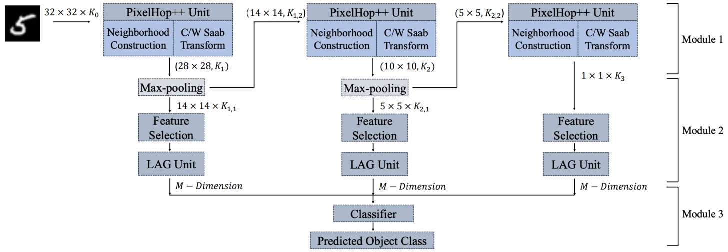
-- Choose *Ns* features with lowest cross-entropy

Note: the index of the selected dimensions for testing set should be the same with training set

ii. Do LAG using *lag.py:*

Note: LAG should be trained (fit) only on training set

HW6 P2: CIFAR-10 Classification using SSL



★ Implementation Steps

4. Module 3:
    i. Gather all the outputs of LAG units from different Hops
    ii. For each image, the features from different Hops are concatenated into a long feature vector
    iii. Choose a classifier (e.g. Random Forest, SVM, Linear Regression, etc)
    iv. Train the classifier on the concatenated feature vector of training set
    v. Test on the testing set

HW6 P2: CIFAR-10 Classification using SSL

Parameters

Table 1 Choice of hyper-parameters of PixelHop++ model for this section

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)


Q
A