EE569 Digital Image Processing

HOMEWORK #6

Issued: 04/14/2022 Due: 11:59PM, 05/01/2022

Problem 1: Origin of Green Learning (GL) (35%)

1.1 Motivation

Nowadays, the development of language models has progressed greatly, but there is a great need for large amounts of data and large size models to obtain better performance. The cost incurred is beyond the reach of most researchers and can cause environmental problems by creating large amounts of CO2 emissions. The aim of green learning is to make our models more environmentally friendly in terms of getting new or better results, to make the amount of computation used lower, or to get better results without increasing the amount of computation. It proposes to let everyone explore another possibility besides increasing data and increasing models.

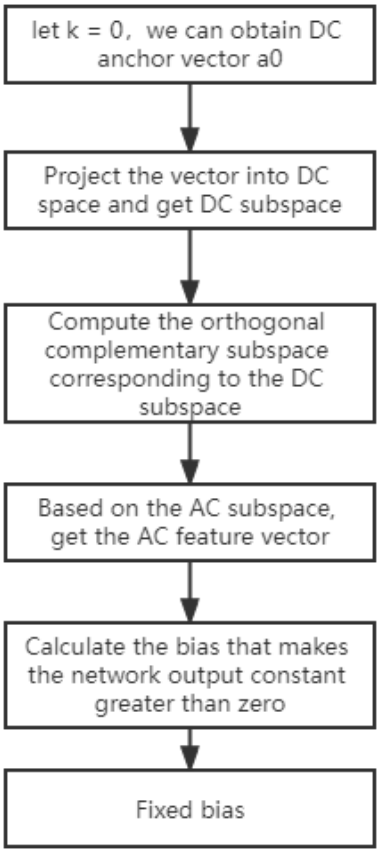
1.2 Approach

The problem is to read the four papers required in the question, understand the Saak transform, the Saab transform, PixelHop and PixelHop++, and answer the related questions.

1.3 Results

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

(1) The flow diagram is as below:



Principal component analysis (PCA) is a commonly used method for subspace dimensionality reduction within the field of machine learning; in fact, the Saab transform is a special kind of principal component analysis method. For an input space of $\mathrm{X} = (\mathrm{x}_0, \mathrm{x}_1, \ldots, \mathrm{x}_{\mathrm{N}-1})^{\mathrm{T}}$, the one-stage Saab transform can be expressed as:

$$y_k = \sum_{n=0}^{N-1} a_{k,n} x_n + b_k = a_k^T X + b_k, k = 0, \ldots, K-1$$

N denotes the dimensionality of the convolution kernel. Projecting the input space onto the DC convolution kernel yields the DC feature space $\mathrm{X}_{\mathrm{DC}} = 1/\sqrt{\mathrm{N}}(\mathrm{x}_0, \mathrm{x}_1, \ldots, \mathrm{x}_{\mathrm{N}-1})^{\mathrm{T}}$, from which AC can be calculated the feature space $\mathrm{X}_{\mathrm{AC}} = \mathrm{X} - \mathrm{X}_{\mathrm{DC}}$. When $\mathrm{k} > 0$, the convolution kernel a_k is called the AC (alternating current) convolution kernel. To obtain the AC convolution kernel, principal component analysis is used to decompose AC, and then $K-1$ feature vectors are selected as the AC convolution $\mathrm{a_k}, \mathrm{k} = 1, \ldots, \mathrm{K}-1$.

(2) Similarity: Both contain convolutional and fully connected layers, and both require weight and bias calculations and updates.

Differences: The FF-CNN replaces convolution by the Saab transform and constructs fully connected layers by clustering and least squares regression algorithms. Signal-

based perspective decomposition and calculation of correlation between input features and output without backward propagation of gradients.

(b) Understanding PixelHop and PixelHop++

(1) Successive subspace learning consists of four parts:

1) Successive near and far neighborhood expansion
2) Unsupervised dimension reduction is achieved through subspace approximation
3) Dimension reduction through label-assisted regression supervision
4) Feature linking and decision making.

Successive subspace learning differs from traditional deep learning in that deep learning directly learns features in a high-dimensional space, while Successive subspace learning learns features in a low-dimensional feature subspace after dimensionality reduction in a high-dimensional space. At the same time, different from the traditional dimension reduction method, the feature subspace is obtained by unsupervised reduction based on subspace approximation and label assisted regression.

(2)

Module 1: Feature extraction

Module 2: Feature dimension reduction
Module 3: Feature linking and decision making

(3)

Both PixelHop and PixelHop++ are image classification algorithms. The neighbourhood is constructed by clipping a certain window size of image patches and keeping a large number of image patches for further Saab transform or the channel-wise Saab transform. PixelHop and PixelHop++ both contain multiple Hop units to extract features from small receptive domains to large receptive domains.

PixelHop++ uses a new feature representation based on tree decomposition, which sorts the features of leaf nodes according to their cross-entropy values and uses them to select feature subsets. At the same time, it uses two separable tensor products to approximate express Saab transformation, which greatly reduces the memory required by Saab transformation.

Channel-wise Saab transform adopts K channels, each Channel inputs S*S image, while Saab transform inputs K*S*S image. Channel-wise Saab transform can greatly accelerate the operation speed through parallel calculation.

1.4 Discussion

Compared with PixelHop, PixelHop++ need less memory, but the accuracy of PixelHop++ may be less than PixelHop.

Problem 2: PixelHop & PixelHop++ for Image Classification (65%)

2.1 Motivation

Both PixelHop and PixelHop++ are image classification algorithms. The neighbourhood is constructed by clipping a certain window size of image patches and keeping a large number of image patches for further Saab transform or the channel-wise Saab transform. PixelHop and PixelHop++ both contain multiple Hop units to extract features from small receptive domains to large receptive domains.

2.2 Approach

(a) Build PixelHop++ model:

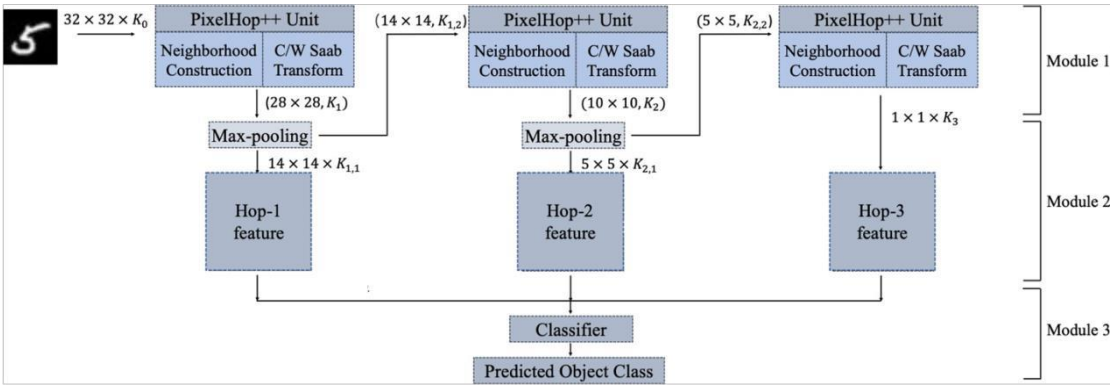
Step 1. Create a function to shuffle data.

Step 2. Create a function to select equal number of images from each classes.

Step 3. Create a function to do shrink operation.

Step 4. Create a function to concate features from different hops and get features.

Step 5. In main function, load dataset, process data, use 10000 images to train PixelHop++. Train Module 1, Module 2, Module 3.



(b) Build PixelHop model:

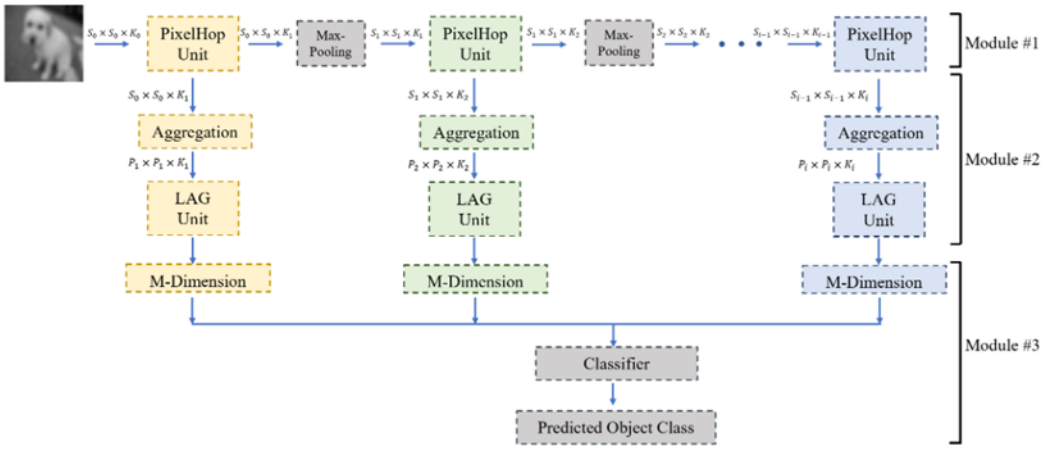
Step 1. Create a function to shuffle data.

Step 2. Create a function to select equal number of images from each classes.

Step 3. Create a function to do shrink operation.

Step 4. Create a function to concate features from different hops and get features.

Step 5. In main function, load dataset, process data, use 10000 images to train PixelHop. Train Module 1, Module 2, Module 3.



(c) Error analysis

Step 1. Load feature data got from main.py

Step 2. Compute the confusion matrix and plot the accuracy curve.

Step 3. Show all the class groups.

		Predicted Class: Spam	Predicted Class: Non-Spam
Actual Class	Spam	TP=45	FN=20
	Non-Spam	FP=5	TN=30

2.3 Results

(a)

(1) & (2)

Table1. Accuracy of PixelHop++ model

	MNIST	Fashion-MNIST
Train Accuracy	0.9858	0.9088
Test Accuracy	0.9652	0.8569

We used (window size) * (window size) * (total features) to calculate the size of the

model.

For the MNIST dataset, the model I constructed yielded 24, 111, and 126 features on hop1, hop2, and hop3, respectively. Considering that there are a total of 261 features in the MNIST dataset. Therefore, the total trainable parameters are (24+111+126) *5*5=6525. For training time, I get 229.4s, 143.3s and 166.4s on module1, module2 and module3 respectively. therefore, the total training time is 539.1s.

For fashion-MNIST, I got 23, 59, 70 features for hop1, hop2 and hop3 unit respectively. Hence, there are 152 features in total and the total trainable parameters is 152*5*5=3800. For training time, I got 180.5s, 108.3s and 93.4s for module1, module2 and module3 respectively. Hence, the total training time was 382.2s.

(3)
For MNIST dataset:

Table2. PixelHop++ MNIST experiment results

TH1	**Test Accuracy**	**Feature Size (hop1, hop2, hop3)**	**Number of parameters**
0.001	0.9680	(24, 111, 130)	6625
0.002	0.9680	(24, 111, 130)	6625
0.004	0.9648	(24, 111, 127)	6550
0.005	0.9636	(24, 111, 126)	6525
0.006	0.9638	(24, 111, 122)	6425
0.008	0.9620	(24, 110, 116)	6250
0.01	0.9629	(24, 108, 108)	6000

Curve of PixelHop++ test accuracy:

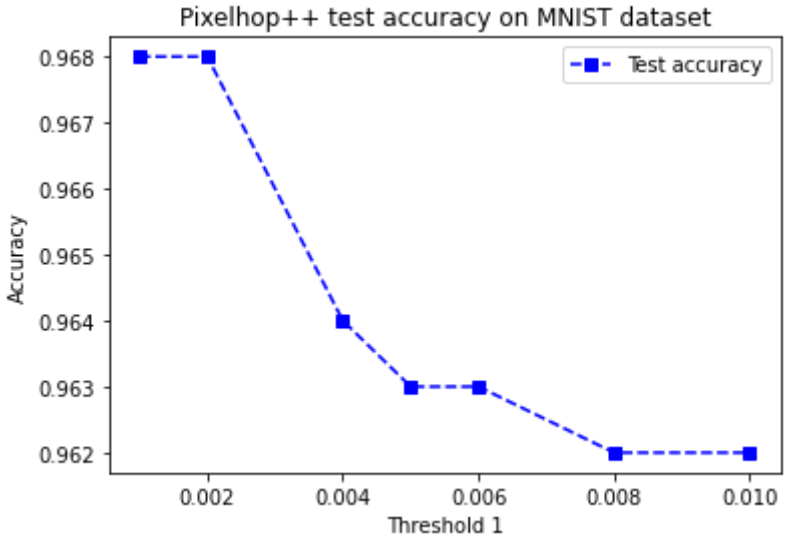


Figure 1: PixelHop++ test accuracy on MNIST dataset

For Fashion-MNIST datasets:

Table3. PixelHop++ Fashion-MNIST experiment results

TH1	Test Accuracy	Feature Size (hop1, hop2, hop3)	Number of parameters
0.001	0.8583	(23, 59, 78)	4000
0.002	0.8583	(23, 59, 78)	4000
0.004	0.8569	(23, 59, 71)	3825
0.005	0.8563	(23, 59, 70)	3800
0.006	0.8516	(23, 59, 62)	3600
0.008	0.8453	(23, 55, 55)	3325
0.01	0.8445	(23, 55, 53)	3275

Curve of PixelHop++ test accuracy:

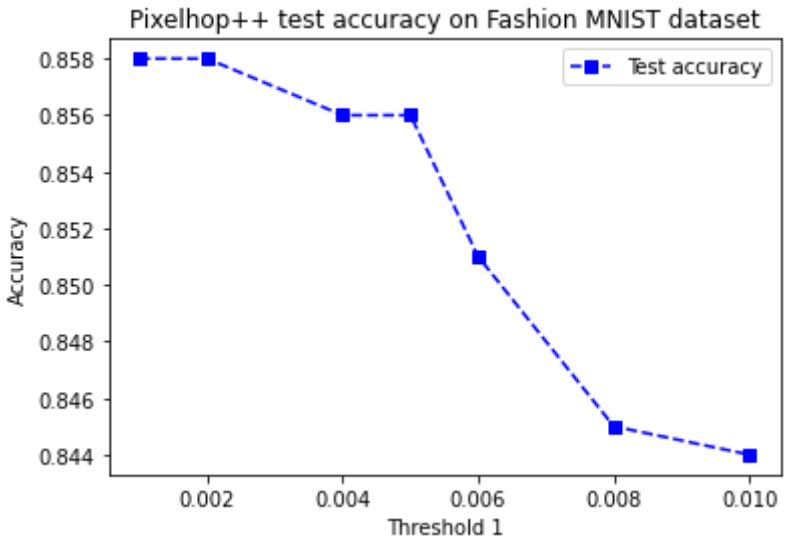


Figure 2: PixelHop++ test accuracy on Fashion-MNIST dataset

From the figure1, figure2, table2 and table3, we can find that with increasing the TH1, there will be less feature from Hop2 and Hop3 preserved, while Hop1 feature doesn't change. Hence. The training time, training accuracy and model size decrease because of less feature will be used.

(b)
(1) & (2)
PixelHop needs more memory and my computer cannot handle it. Here I used 10000 balanced data from training dataset. Besides, during getting feature process, I sliced the training data to 6 subsets with 10000 data each and extracted feature iteratively by batch size. To be fairness, I used the same data strategy for PixelHop++ for comparison. Here I listed the performance of two algorithms with the default settings. Here I train the

PixelHop and PixelHop++ model with the same setting default settings:

For PixelHop model size:
Hop size = window size * window size * number of former hop feature * number of this hop feature

For PixelHop++ model size:
Hop size = window size * window size * number of hop feature

Table3. Training Accuracy of PixelHop and PixelHop++

TH2	MNIST		Fashion MNIST	
	Pixelhop	Pixelhop++	Pixelhop	Pixelhop++
0.001	0.9874	0.9873	0.9122	0.9135
0.002	0.9838	0.9778	0.9026	0.8897
0.005	0.9748	0.9395	0.8701	0.8567
0.01	0.9661	0.8882	0.8421	0.8039
0.015	0.9565	0.8421	0.8234	0.7724

Table4. Performance on MNIST vs TH2

TH2	PixelHop Model			PixelHop++ Model		
	Test Accuracy	Feature Size (hop1, hop2, hop3)	Number of parameters	Test Accuracy	Feature Size (hop1, hop2, hop3)	Number of parameters
0.001	0.9664	(24,57,69)	133125	0.9689	(24,109,131)	6600
0.002	0.9610	(20,40,41)	61500	0.9555	(20,66,59)	3625
0.005	0.9536	(13,24,24)	22525	0.9091	(13,33,21)	1675
0.01	0.9455	(10,16,16)	10650	0.8458	(10,17,10)	925
0.015	0.9311	(8,12,13)	6500	0.8062	(8,12,7)	675

Table5. Performance on Fashion-MNIST vs TH2

TH2	PixelHop Model			PixelHop++ Model		
	Test Accuracy	Feature Size (hop1, hop2, hop3)	Number of parameters	Test Accuracy	Feature Size (hop1, hop2, hop3)	Number of parameters
0.001	0.8611	(23,44,42)	72075	0.8586	(23,58,78)	3975
0.002	0.8530	(18,28,29)	33350	0.8415	(18,37,43)	2450
0.005	0.8282	(12,16,15)	11100	0.8092	(12,19,20)	1275
0.01	0.8072	(8,11,10)	5150	0.7632	(8,11,12)	775
0.015	0.7890	(6,8,8)	2950	0.7402	(6,8,7)	525

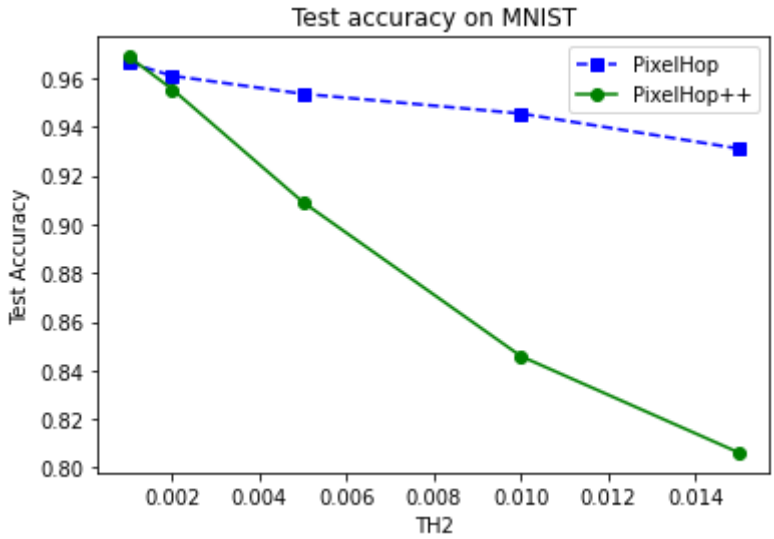


Figure 3: Test accuracy of PixelHop and PixelHop++ on MNIST

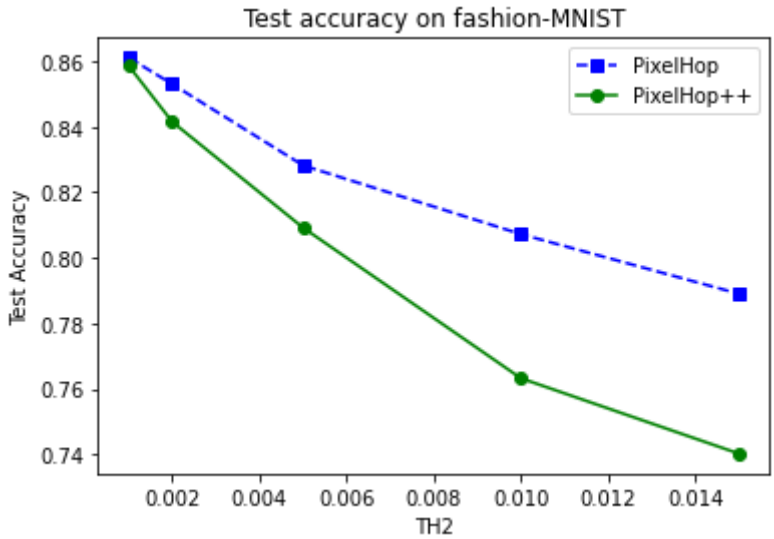


Figure 4: Test accuracy of PixelHop and PixelHop++ on Fashion-MNIST

From Table4. and Table5, we can find that Pixelhop has more parameters because they use traditional convolution has element-wise add, since they use multiple kernels to compute for one feature maps, while in PixelHop++ only one feature map contribute to the next step, they don't use element-wise add.

For TH2, the accuracy and model size decreases with increase of TH2. When TH2 is low, the accuracy of PixelHop ++ is really closed to PixelHop, while if increase the TH2 PixelHop ++ suffer more. TH2 suffer more is because there is a TH1 in Pixelhop++. If the two value didn't corporate clearly, we would discard high energy features and pertain lower energy features. For model size, the difference from PixelHop and PixelHop ++ decrease with increases of TH2, shown as figure3.

Besides, when the two model have similar parameters, the PixelHop ++ have higher accuracy than PixelHop.

(c)
(1)
For MNIST datasets:

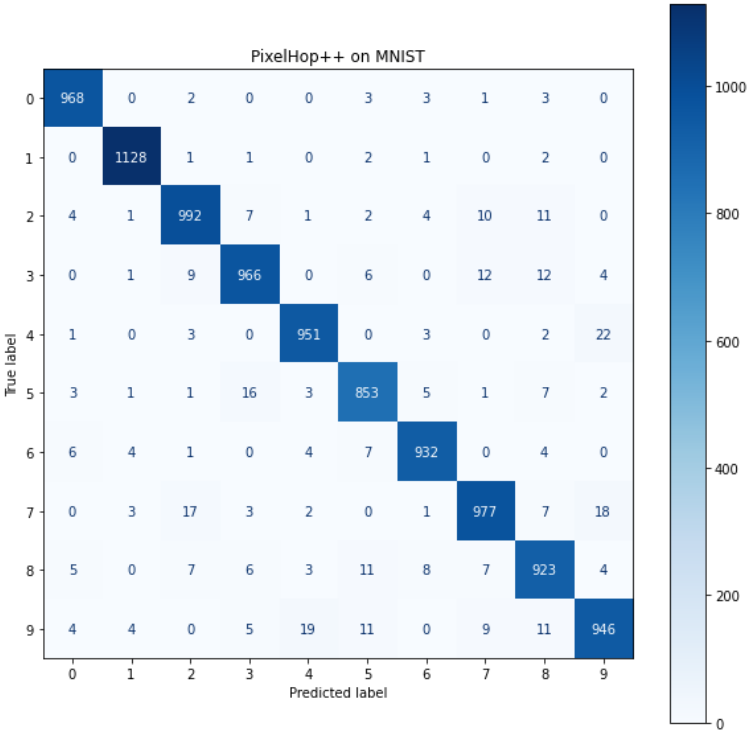


Figure 5: Confusion matrix on MNIST

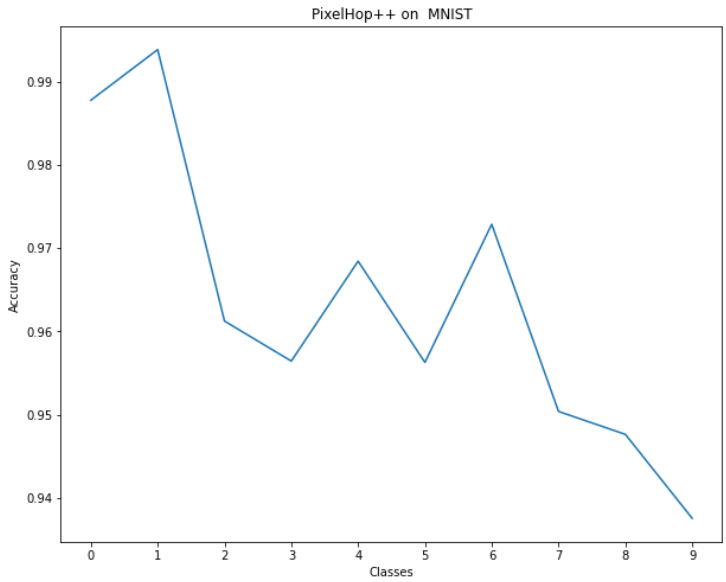


Figure 6: Test accuracy verse class on MNIST

From the confusion matrix heat map and accuracy curves verse class, we can find that class “9” has the lowest accuracy and class “1” has the highest accuracy, which means class “9” has the highest error rate and class “1” has the lowest error rate. According to common sense, “1” is the easiest class because it is very simple and other digits looks quite different from “1”.

For Fashion-MNIST datasets:

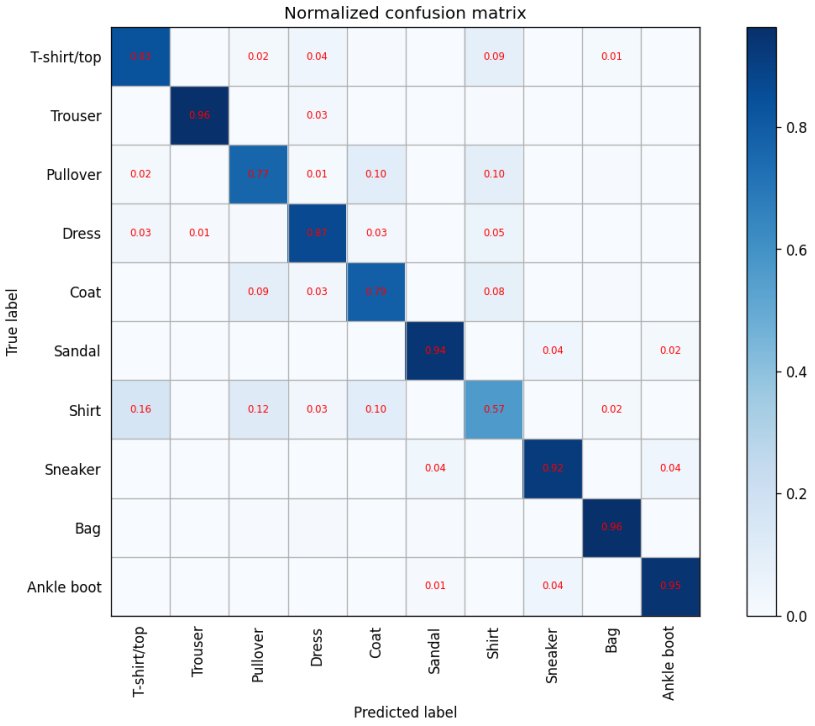


Figure 7: Confusion matrix on Fashion-MNIST

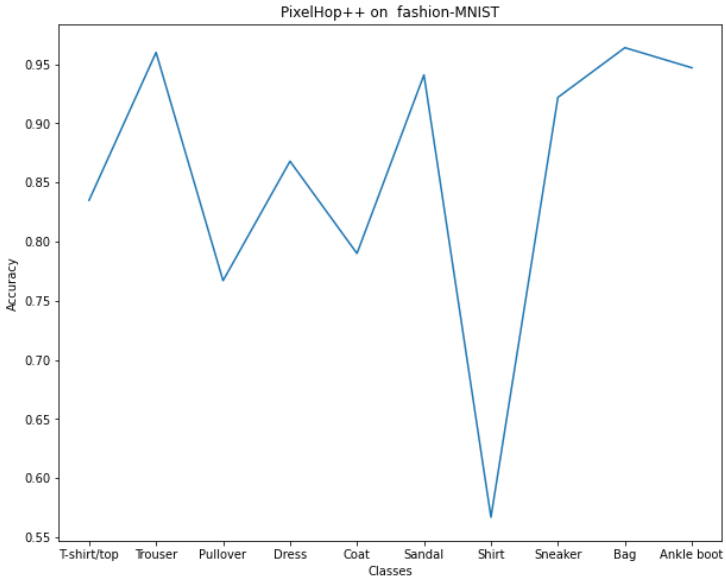


Figure 8: Test accuracy verse class on Fashion-MNIST

(2)
For MNIST datasets:

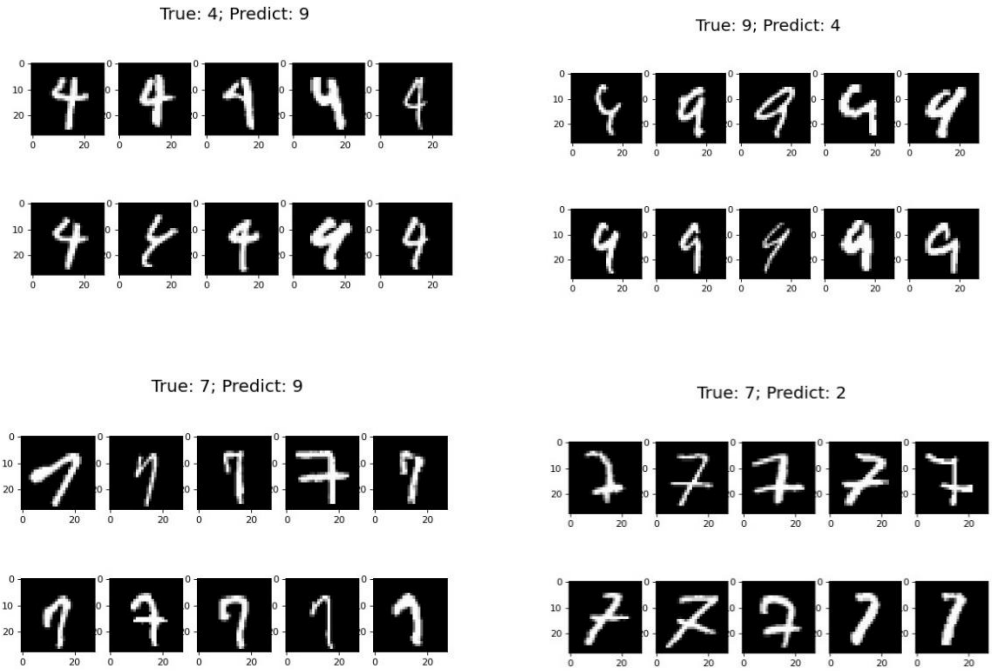
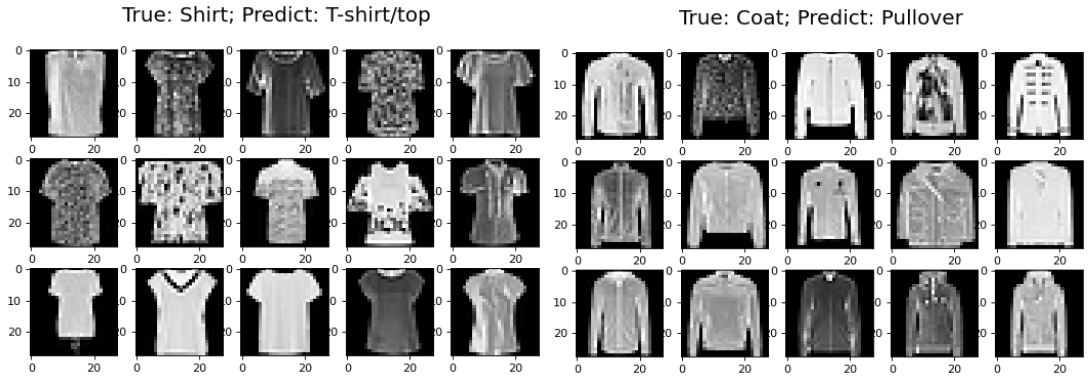


Figure 9: Confusing class groups on MNIST

From the confusion matrix, we can find that “9” is easy to get confused with “7” and “4” and “7” is easy to get confused with “2”, as shows above. Here I plot the images that model get confused between “4” and “9”. I think humans cannot classify these images in total either. In the common sense, the upper part of “4” is unclosed while the upper part of “9” is closed in hand-written digits. However, in the data plot above “4” and “9” look quite similar and some of images with annotated as “9” are also unclosed, or vice versa. For images that miss classifying 7 as 9, the upper part of 7 digits is going to close, which is the feature of “9”. Hence, it is easy to misclassify. The same scenarios happened in “7” and “2”, some digit “7” have a line horizontally which makes them looks like “2”.

For Fashion-MNIST datasets:



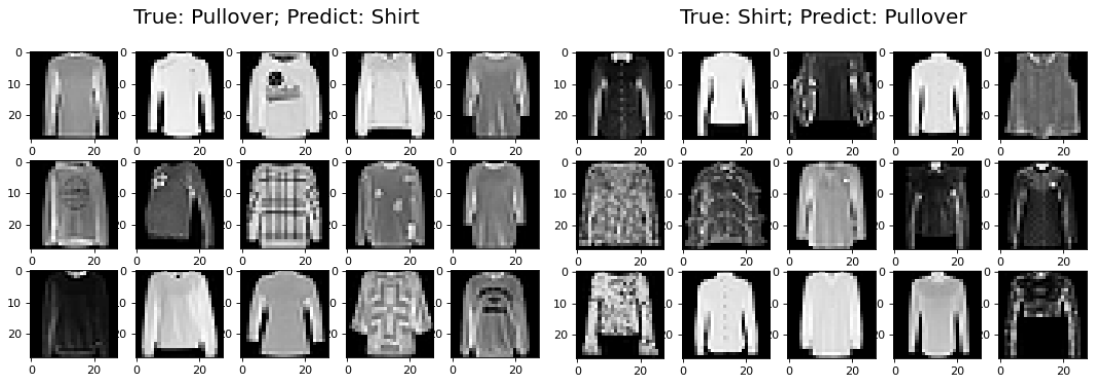


Figure 10: Confusing class groups on Fashion-MNIST

From the Confusing class groups on Fashion-MNIST datasets above, we can find that Pullover, shirt and T-shirt looks very similar. Actually, I can classify them by my eyes. The difference between the pullover and shirt is whether it has button. However, in the image shows above, there are multiple reasons for miss-classification. The first is that the images in Fashion-MNIST are gray images and the image size is 28 by 28 pixels, so it cannot recover enough information. Besides, there are some shape or texture information on the shirts, which will cover other information, like buttons.

(3)
In the PixelHop++, we used PCA to assign the convolutional network parameters. However, PCA is an unsupervised method, which means that in the PixelHop ++ units, we don't use the label information. Besides, In PixelHop ++, we used a tree-based model. The original decision tree-based model use label to guide which nodes to choose. However, In the PixelHop ++, we only use energy. Here, I think if we use the label assistant energy, it will get the higher results. Here, I think we can modify that criterion, using class-wise energy. Hence, we can change our criterion to discriminant power, like Linear Discriminant Analysis (LDA). We need to find the features that have high inter-class (between different class) distance and low intra-class distance (within the class). These features benefit the target a lot.

Besides, according to the test results, some images cannot be classified because that they cannot be viewed clearly. Hence, I think data augmentation could also have some improvement on the result.

2.4 Discussion
From the above experiments we can find that the accuracy of PixelHop is better than PixelHop++. We can find that PixelHop has more parameters because they use traditional convolution has element-wise add, since they use multiple kernels to compute for one feature maps, while in PixelHop++ only one feature map contributes to the next step, they don't use element-wise add.

For TH2, the accuracy and model size decreases with increase of TH2. When TH2 is

low, the accuracy of PixelHop++ is really closed to PixelHop, while if increase the TH2 PixelHop++ suffer more. We should note that in reality, we should not try TH2 > TH1. Here, I try some scenarios that TH2 > TH1 because I want to compare the two algorithms. TH2 suffer more is because there is a TH1 in PixelHop++. If the two value didn't corporate clearly, we would discard high energy features and pertain lower energy features. For model size, the difference from PixelHop and PixelHop++ decrease with increases of TH2.