HW6

Problem1. Origin of Green Learning

(a) FF-CNN

(1) Summarize Saab Transform

Saab transform is a method based on Principle Component Analysis. In the kernel training process, the Saab transform use PCA to extract information dataset and adding bias. This method can be used to extract AC and DC anchor vector, which can be used in further convolution neural network. Compared with the traditional PCA method, Saab transform introduced the bias that is DC part a $1 \times length$ one vector. After adding the bias, all output can be positive, which can reduce the non-linear activation function, ReLU. Compared with Saak transform, Saak transform used positive and negative anchor vector pairs instead of adding the bias. Hence Saak transform only preserve half of output after ReLU, while Saab transform can preserve all vectors.

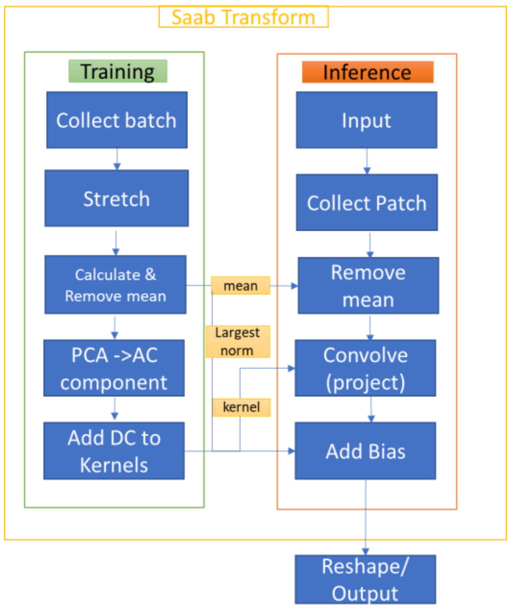


Figure1. Saab Transform Flowchart

Above Flowchart is extracted from the code. The first step is to extract the kernel weights. During the convolution, every pixel will be sum up in special windows, Hence the first step is to extract window size pixels and arrange them. After that, remove the bias and Apply PCA to find

the parameters for convolution kernels, which is considered as DC part. Combine DC with AC and then we get the anchor vectors.
During the inference part, first, add the bias if needed. Then, removing the mean calculated in training processing and convolve the input with kernels obtained during the training. The output of Saab transform is the energy loss and feature maps. Which can be evaluate the performance and using for classification or other problems.

(2) Similarity and Difference

Similarity:
Both methods are the way to get the parameters of CNN networks from training data. Both networks have the same pipeline. They use convolution kernels to do the feature extraction. Hence, they are similar during the inference on feature convolution part.

Difference：
1) FF-CNN don’t need iterations during feature extraction, while BP need multiple iterations to decide which to calculate the parameters, which time and computation consuming.
2) BP- CNN can use backward propagation to determine the parameters not only for CNN part but also for fully connected layer. Hence, this method is end-to-end pipeline. FF-CNN can only use Saab transform to calculate the parameters for CNN part. The FC layers the author use the LAG to determine that part of parameters, which is not end-to-end. Hence FF-CNN have multiple goodness. They are modularized so they are easy to debug and tune the parameters of the models. Besides, FF-CNN can be explained mathematically so they features can be used in other methods.
3) During the training process, FF-CNN need to determine each layer’s parameters cascade, while BP-CNN can train the whole network in the same times.
4) FF-CNN is time and computational efficient which is regarded as green learning. Which BP-CNN need to calculate all nodes’ gradients and iterate multiple times to search for the best parameters, which is time and computational consuming.
5) FF-CNN is less randomness while BP-CNN have more randomness because of the different patches. For BP-CNN during the training process, the parameters may trap into local minimum which will influence the performance.
6) Hence, BP-CNN is more reliable on the dataset. Large dataset may increase the performance. FF-CNN don’t rely on too much the dataset and hence it is more robust and stable track.

(b) PixelHop and PixelHop++

(1) Explain SSL

Successive Subspace Learning is a new-track method that use the statistics of data to determine the parameters of the pipeline. The traditional successive subspace learning contains multi-stage feature extraction, dimension reduction with pooing, label assistant regression and feature concatenation and classification. Successive subspace learning is combining traditional machine learning algorithms, mathematic and statistic method with CNN. They use saab transform to calculate the parameters of convolution neural network, use CNN to extract features and use features to train the aggregation and LAG to reduce the dimension of feature and finally use concatenation and classifier to make prediction.

Deep learning algorithm need back propagation to train the network and the process is considered as an optimization problem, which is an end-to-end problem. SSL methods use the dataset statistics. Hence, it is more explainable and easier to implement because it can be regarded as Convex optimization problem while the back propagation is optimization problem over the whole space and hard to find global optimized point. SSL doesn't need iterations architecture. Subspace successive learning can be divided into three modules which means that it can be modularized and use the intermediate output which is easier to debug or transfer learning. Besides SSL can directedly implement on CPU with less time consuming and computational consuming while deep learning needs to implement the code onto GPU for training, which need more time to do iterative optimization and training. Besides, Saab transform in SSL is an unsupervised track, which can reduce the heavy load of data annotations, even though module#2 and module#3 need annotation data. Traditional deep learning method with back propagation need label supervising for sure, needing researchers to annotate data, which need heavy workload.

(2) The functions of Module

The module #1 PixelHop Unit Cascade structure. The function of this part is to extracted features and compute the kernel parameters of the Network. For each PixelHop unit, it calculated the parameters by Saab transform. Then use the spatial pooling to convey the difference to the next layers, which can increase the receptive field. The module #2 is constructed by aggregation and LAG units to reduce the feature map dimension to a M-dimension vector. After each PixelHop Unit, the aggregation transformation which reduce the original feature map (m, m) to (m, m) and calculate the mean, max and minimum values in aggregation and label assistant regression. During the label assistant part, use Cluster to divide the data of same labels into different part. Then assign different part with pseudo labels. Finally, do label assistant regression to compute M-dimension vector for every LAG Units, which regards as dimension reduction part. The module #3 is feature concatenation layers which use the classifier to do classification by M-dimension vectors from different LAG units.

(3) Comparison between pixelhop and pixelhop++

Both pixelhop and pixelhop++ can be regarded as image classification algorithm. The neighborhood construction is to clip a certain window size image patches and preserve the plenty of image patches for Saab transform in further. Pixelhop and Pixelhop++ both contains multiple Hop unit to extract features from small receptive field to high receptive field. Subspace approximation part is to use Saab transform in Pixelhop and channel-wise Saab transform in

Pixelhop++ to extract the features and determine the kernel parameters' method. Saab transform only have one threshold to determine where the feature can be used or not, while channel-wise saab transform have double threshold to determine whether drop or not and whether leave the nodes to be leaf or not. From the convolution perspective, Pixelhop use traditional convolution, which contains element-wise add and need multiple kernel size to contribute one feature map, hence it need huge memory and have more parameters. For Pixelhop++, channel-wise saab transform only need one kernel to calculate one feature map, which need less parameters. The high threshold determines where the feature need to convey to next layer. The features with higher threshold will convey to next layer for cascade element wise Saab, while others between the two thresholds will be left as leaves. The whole structure of Pixelhop++ looks like a tree. Pixelhop++ is a low complexity method. The number of parameters is lower because of dropping features and leaving features as leaf. In reality, computer need more memory for pixelhop and my pixelhop program will crash while pixelhop++ would not in the same setting of MNIST dataset (training the whole dataset).

Problem2. Classification of MNIST and Fashion-MNIST dataset

(a) PixelHop++

(1)&(2) training time, training accuracy, test accuracy

My computer's CPU is I7-7700 with 16G Memory. Besides, I also have GPU Nvidia GTX 1070Ti, which can help me save a bunch of time during the XGBOOST model training process.

For Pixelhop++ model, I trained module 1 with the whole training dataset. After that, I use function get_feat() for the whole training set and test set without Batch to get Hop3 feature. Here is the result under default settings. The model Size is Calculated by the **window size * window size * total Features**.

Table1. Accuracy of PixelHop++ model

	MNIST	**Fashion-MNIST**
Training Accuracy	0.9858	0.9073
Test Accuracy	0.9653	0.8553

For MNIST, I got **24, 111,126** features for hop1, hop2 and hop3 respectively. In MNIST dataset, there were 261 filters in total. Hence, the total trainable parameters are (24+111+126) * 5*5 = **6525**. For training time, I got **375.4s, 242.4s** and **162.19s** for module1, module2 and module3 respectively. Hence, the total training time was 780.0s.

For fashion-MNIST, I got **23, 59, 70** features for hop1, hop2 and hop3 unit respectively. Hence, there are 152 filters in total and the total trainable parameters is **3800**. For training time, I got **187s, 92.6s and 90.17s** for module1, module2 and module3 respectively. Hence, the total training time was 397s.

(3) TH1 Verses PixelHop++

Here I tried different TH1 to test the value of TH1 influence, other settings follow as the table in problem descriptions. The model Size is Calculated by the **window size * window size * total Features**.

Note: The large-scale decrease is because I use large TH1 range to 0.14.

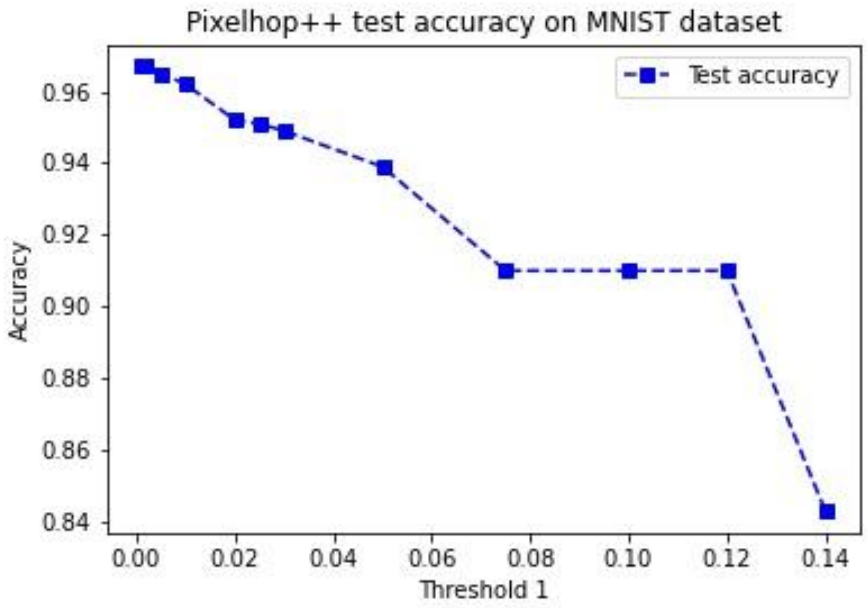


Figure2. Test accuracy of PIxelHop++ on MNIST dataset

Table2. Pixelhop++ MNIST experiment results

TH1	Test Accuracy	Feature Size (hop1, hop2, hop3)	Number of parameters
0.001	0.9675	(24, 111, 130)	6625
0.0015	0.9675	(24, 111, 130)	6625
0.002	0.9675	(24, 111, 130)	6625
0.005	0.9653	(24, 111, 126)	6525
0.01	0.9623	(24, 108, 108)	6000
0.02	0.9519	(24, 89, 58)	4275
0.025	0.9519	(24, 89, 58)	4275
0.03	0.9398	(24, 89, 51)	4100
0.05	0.9398	(24, 65, 34)	3075
0.075	0.9109	(24, 47, 22)	2325

0.1	0.9109	(24, 47, 22)	2325
0.12	0.9109	(24, 30, 22)	1900
0.14	0.8431	(24, 30, 9)	1575

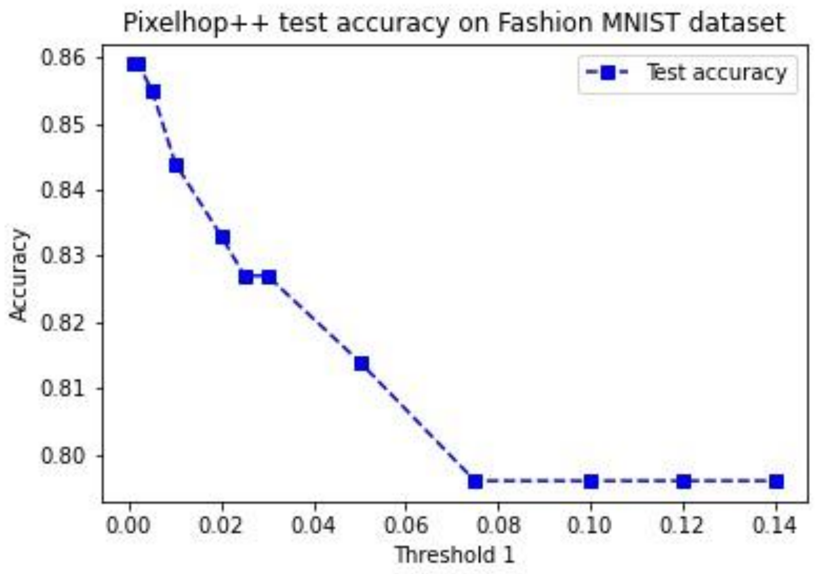


Figure3. Test accuracy of PIxelHop++ on Fashion-MNIST dataset

Table3. Pixelhop++ Fashion-MNIST experiment results

TH1	Test Accuracy	Feature Size (hop1, hop2, hop3)	Number of parameters
0.001	0.8593	(23, 59, 78)	4000
0.0015	0.8593	(23, 59, 78)	4000
0.002	0.8593	(23, 59, 78)	4000
0.005	0.8553	(23, 59, 70)	3800
0.01	0.8445	(23, 55, 53)	3275
0.02	0.8331	(23, 40, 32)	2375
0.025	0. 8279	(23, 35, 28)	2150
0.03	0. 8279	(23, 35, 28)	2150
0.05	0.8142	(23, 28, 23)	1825
0.075	0.7965	(23, 20, 16)	1475
0.1	0. 7965	(23, 20, 16)	1475
0.12	0. 7965	(23, 10, 16)	1225
0.14	0. 7965	(23, 10, 16)	1225

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 and training accuracy decreased because of less feature will be used.

(b) Compare Pixelhop with Pixelhop++

Pixelhop need 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 * # of former hop feature * # of this hop feature

For PixelHop++ model size:

Hop size = window size * window size * # of hop feature

Table3. Training Accuracy of Pixelhop++ and Pixelhop

TH2	**MNIST**		**Fashion MNIST**	
	Pixelhop	**Pixelhop++**	**Pixelhop**	**Pixelhop++**
0.001	0.9878	0.9856	0.9123	0.9086
0.002	0.9835	0.9774	0.902	0.8889
0.005	0.9747	0.9393	0.87	0.8569
0.01	0.9667	0.8872	0.8430	0.8031
0.015	0.9562	0.8416	0.8235	0.7731

Table5. Performance on MNIST vs TH2

	PixelHop Model			**PixelHop++ Model**		
TH2	**Test Accuracy**	**Feature Size (hop1, hop2, hop3)**	**Number of parameters**	**Test Accuracy**	**Feature Size (hop1, hop2, hop3)**	**Number of parameters**
0.001	0.9662	(24,57,69)	133125	0.965	(24,109,128)	6525
0.002	0.9596	(20,40,41)	61500	0.9556	(20,66,59)	3625
0.005	0.9538	(13,24,24)	22525	0.9084	(13,33,21)	1675
0.01	0.9457	(10,16,16)	10650	0.8455	(10,17,10)	925
0.015	0.9307	(8,12,13)	6500	0.8038	(8,12,7)	675

Table6. 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	0.8625	(23,44,42)	72075	0.8583	(23,58,70)	3775
0.002	0.8542	(18,28,29)	33350	0.8405	(18,37,42)	2425
0.005	0.8296	(12,16,15)	11100	0.8096	(12,19,20)	1275
0.01	0.8054	(8,11,10)	5150	0.7632	(8,11,12)	775
0.015	0.7888	(6,8,8)	2950	0.7398	(6,8,7)	525

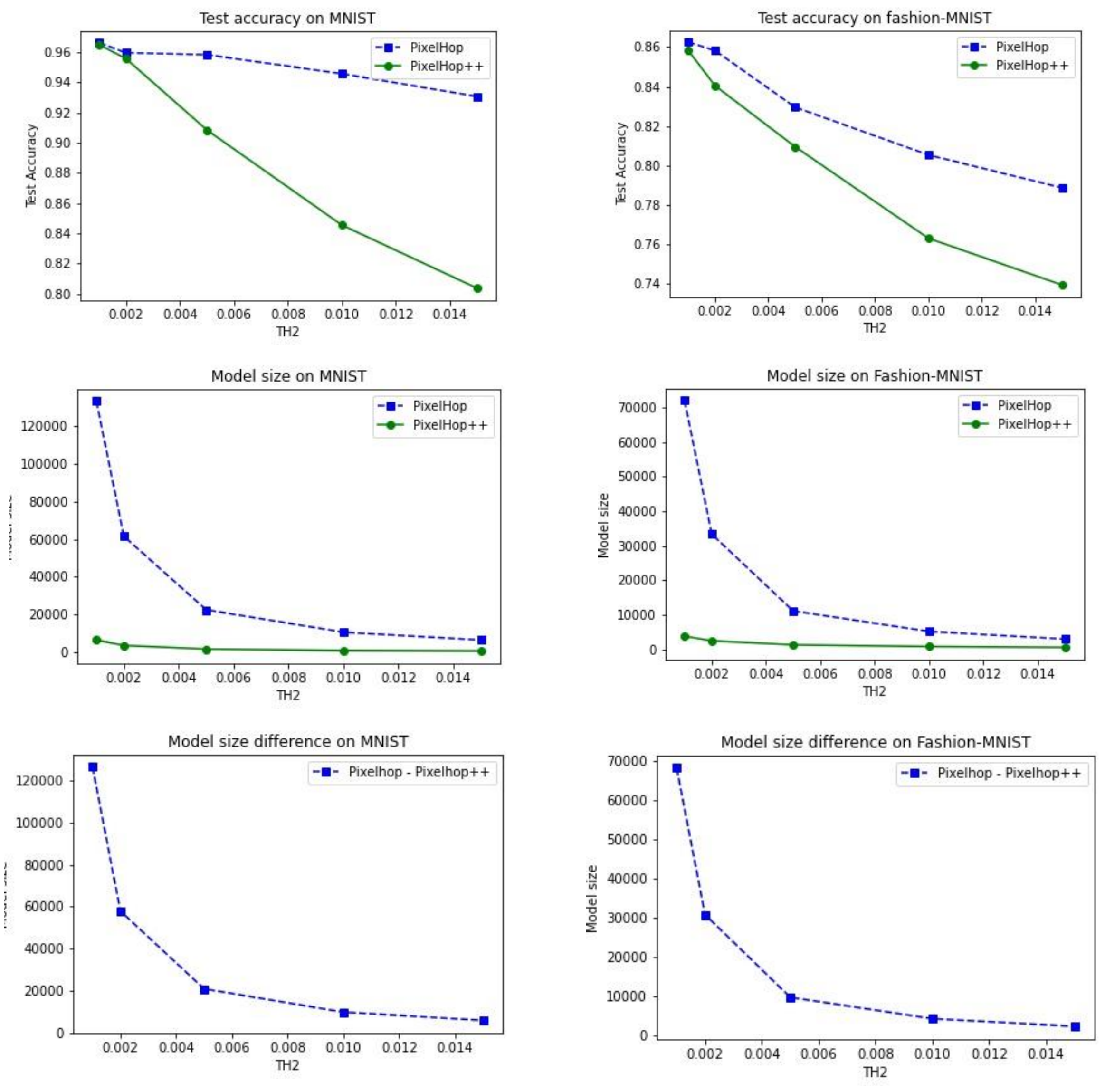


Figure4. Comparison of Pixelhop and PixelHop++ vs TH2

Note: The large difference gap is because I use large TH2 range from0.001 to 0.014.

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. **NOTE: IN REALITY, WE SHOULDN'T 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, shown as figure3.

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

(c) Error analysis

(c1) MNIST dataset

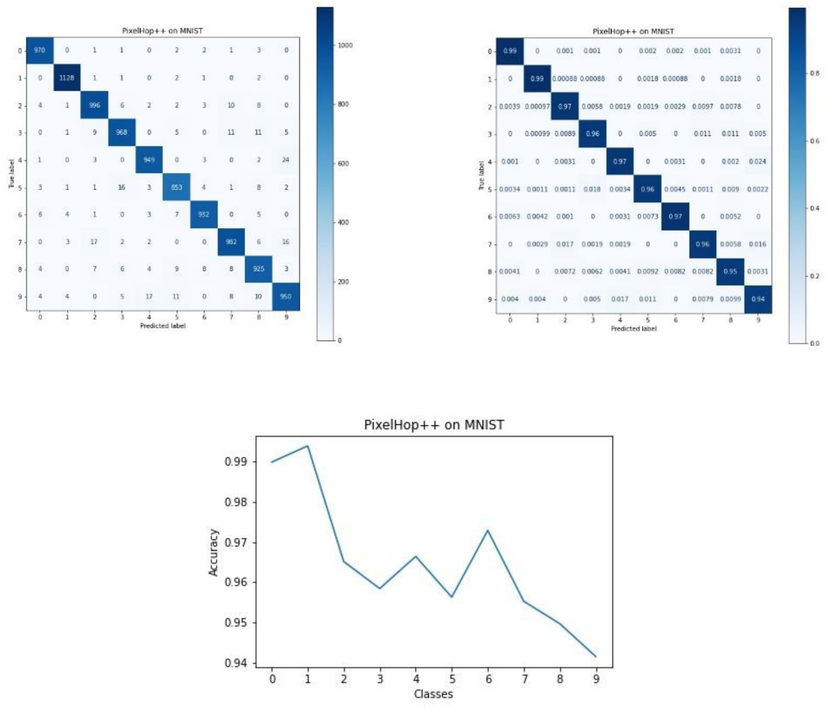


Figure5. Confusion Matrix on MNIST dataset and test accuracy verse class

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

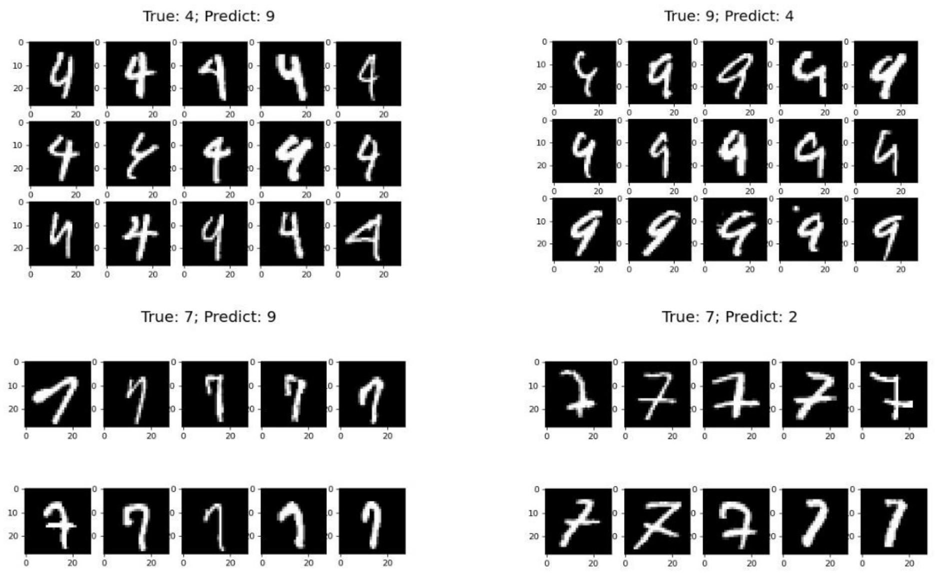


Figure6. Wrong predict sample on MNIST dataset

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”.

(c2) Fashion-MNIST dataset

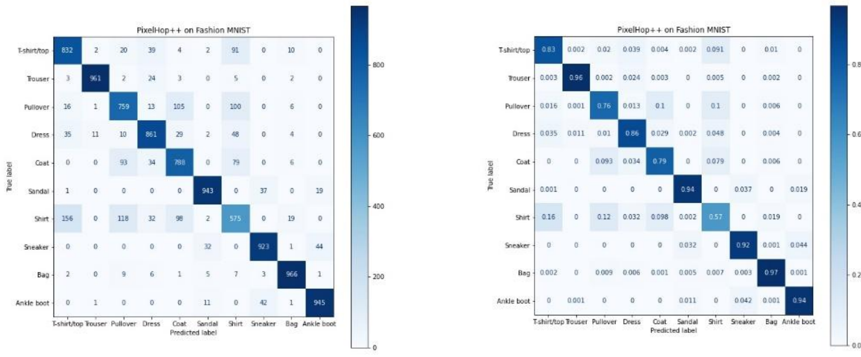


Figure7 Confusion Matrix on Fashion-MNIST datastet

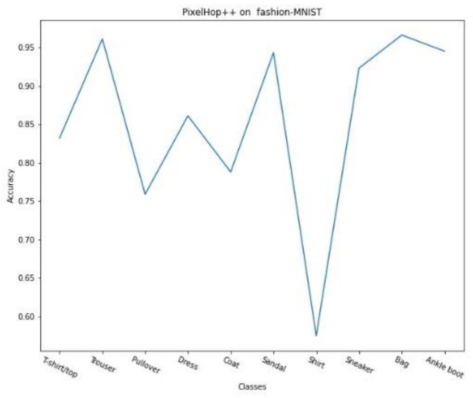
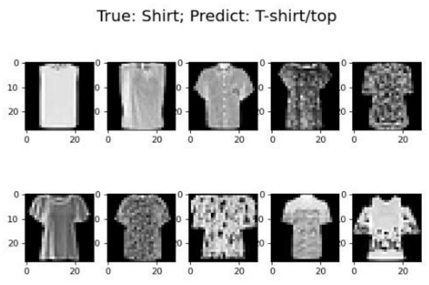
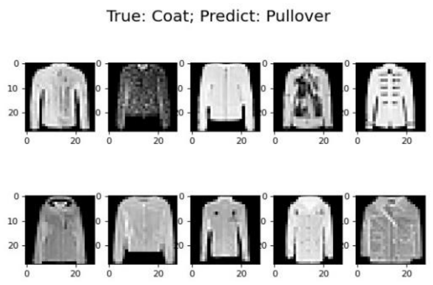


Figure8. Test accuracy Verse Classes

From the Figure of Confusion Matrix map, we can find that the model got worst accuracy on the **"Shirt"** class and the highest accuracy on **"bag".** In common sense, bags are the quite difference with other labels, clothes or shores. Hence, it is quite reasonable for bag to get the highest accuracy.

From the confusion Matrix, we can find that shirt is most likely to confused with T-shirt. Besides, shirts are also easy to get confused with coats and pullovers. Costs are easy to get confused with pullovers.





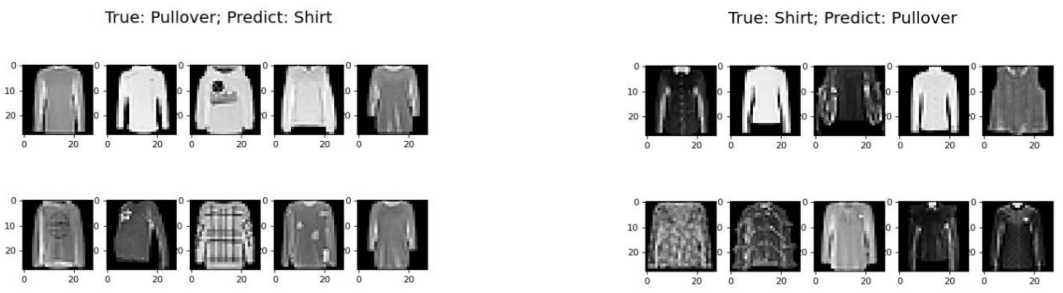

True: Pullover; Predict: Shirt
True: Shirt; Predict: Pullover

Figure9. Wrong Predict sample on Fashion-MNIST dataset

From the wrong predict sample 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.

(c3)

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.