EE569 Introduction to Digital Image Processing

Homework Report #6

Name: Boyang Xiao **USC ID:** 3326730274 **Email:** boyangxi@usc.edu

I. Origin of Green Learning (GL)

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

1.1.1. Non-programming questions

Q1: Summarize the Saab transform with a flow diagram. Explain it in your own words.

Answer:

The flowchart for Saab transform is shown in Figure 1.1 below.

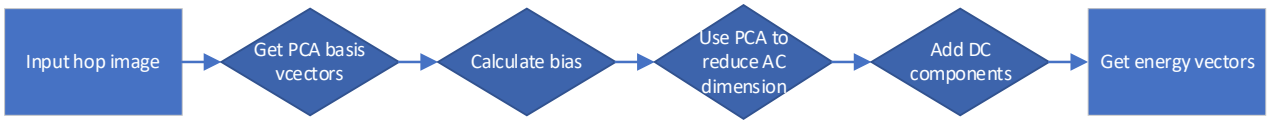


Figure 1.1 Saab transform flow diagram

The Saab transform happens right after the neighboring hop operations. The input hop image is obtained from a hop in the input or the output image. Firstly, Saab transform gets the AC components, which are PCA basis vectors, from the input data. After that, the AC components are used to calculate the bias and PCA is applied to reduce AC vectors' dimensions and get the principal features of the input data. The final step is to add DC components back to the AC vectors and get the final kernels.

Q2: Explain similarities and differences between FF-CNN and backpropagation-designed CNN (BP-CNNs).

Answer: The most essential ***difference*** between FF-CNN and BP-CNN is the way they extract features from the images and train the networks. The main idea of the classic BP-CNN is back-propagation, which means adjusting the weights saved in the fully connected layers (FC layers) to train the networks. As a model especially aimed at the IMAGES, the most important component in a BP-CNN is a series of convolutional layers, which works as a feature extractor and extracts the features from the images. However, FF-CNN uses Saab operations to replace the convolutional operations. FF-CNN is feedforward-designed, and it does NOT require labels or iterations to train the whole networks.

There are also some similarities lying between the FF-CNN and BP-CNN. The main architectures for these two modes are very similar, because they both extract the features from the images and do the classification based on the features extracted.

1.2 Understanding PixelHop and PixelHop++

1.2.1. Non-programming questions

Q1: Explain the SSL methodology in your own words. Compare Deep Learning (DL) and SSL.

Answer: Successive Subspace Learning (SSL) means repeatedly adjusting/training the subspaces of a whole set. The subspace can be obtained from a proportion of the whole set, such as a window of a single image from the whole image dataset. The vectors/kernels of the subspaces will be adjusted to better extract the features from the images.

The difference between SSL and DL is that DL treats the input data as a whole and adjusts the model with the whole bunch of data while SSL does not. SSL partitions the input data into smaller subspaces and train the model with these subspaces iteratively. Based on this idea, DL needs more training samples and labels to adjust the whole model and it is strongly supervised, but SSL does not. SSL requires less training data to achieve the same effects.

Q2: What are the functions of Modules 1, 2 and 3, respectively, in the SSL framework?

Answer: Module 1 is designed to partition the whole images into subspaces and get the feature vectors of the subspaces. This layer does the successive near-to-far neighborhood extension and the features are extracted according to pixels' distance or pixels' energy in the neighborhood. The output will be the multi-dimension feature vectors of the input images.

Module 2 does the feature reduction. PCA is applied to the feature vectors we extracted from the module 1 and the most correlated dimension of the features will be preserved in this module, to generate the feature vectors for classification.

Module 3 is the classification module which does the most essential job in the module: classifying the feature vectors to obtain the predicted classes of the input images. This module in an ordinary module in a classification model and it highly depends on the features extracted by the previous modules.

Q3: Explain the neighborhood construction and subspace approximation steps in the PixelHop unit and the PixelHop++ unit and make a comparison. Specifically, explain the differences between the basic Saab transform and the channel-wise (c/w) Saab transform.

Answer: The neighborhood construction step in the PixelHop unit and the PixelHop++ unit are designed to get the central pixel and the surrounding 8 neighboring pixels to form a sub-vector of the whole image. The subspace approximation step is designed to apply the Saab transform to the sub-vector obtained from the neighborhood construction step.

The differences: Basic Saab transform process all the channels together. For example, if the input image is a gray-scale image, the spatial dimension will be ONE; and if the input image is a RGB image, the spatial dimension will be THREE. But the c/w Saab(channel-wise Saab), as the name suggests, process each channel separately and treat each channel as an input to do the Saab transform.

II. PixelHop & PixelHop++ for Image Classification

2.1. Part (a) Building PixelHop++ Model

2.1.1. Abstract and motivation

In Section I, the basic theories and ideas of PixelHop and PixelHop++ are introduced through answering several questions about these methods. In this part of question, the real PixelHop++ model will be built and trained to test the MNIST dataset and the Fashion-MNIST dataset. The datasets scale will be reduced and the accuracies will be analyzed. Also, different parameters will be tried to show the difference these parameters make to the model.

2.1.2. Approaches and procedures

The PixelHop++ model structure is specifically shown in Figure 2.1 below. The detailed explanations for each module in the PixelHop++ model is elaborated in Section 1.2.1.

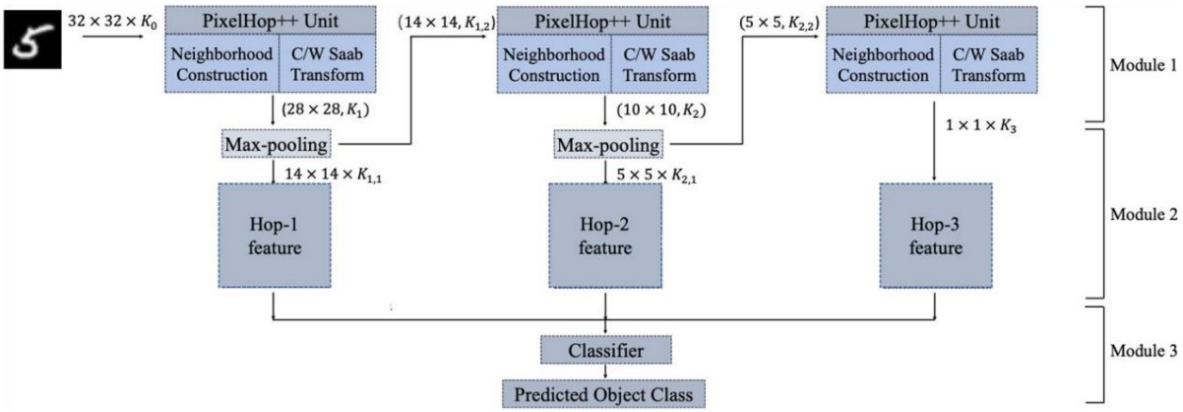


Figure 2.1 PixelHop++ model structure

In this part of question, codes for the Modules components are pulled from the Github repo and the specific training parameters are set to train the model. The initial hyperparameters setting is shown in Table 2.1 below. The training results, including the training accuracies, the training time, the testing accuracies and the total number of parameters will be displayed.

Table 2.1 Initial hyperparameters setting for PixelHop++ model

Spatial Neighborhood size in all PixelHop++ units	5x5
Stride	1
Max-pooling	(2x2) -to- (1x1)
Energy threshold for intermediate nodes (*TH1*)	0.005
Energy threshold for discarded nodes (*TH2*)	0.001
Classifier	XGBoost
Number of estimators in classifier	100

After trying the initial hyper-parameters setting, different *TH1* settings for the model will be tried to train the model and the correlations between the TH1 settings and the testing accuracies will be shown to analyze the influence of TH1 parameter.

2.1.3. Training results with the initial hyper-parameter settings

Using the initial hyper-parameter settings shown in Table 2.1, the models are trained with both the MNIST dataset and the Fashion-MNIST dataset. The training results are shown below in Table 2.2.

Table 2.2: Training results with the initial hyper-parameter settings

Dataset	Training accuracy	Training time	Testing accuracy	Num of parameters
MNIST	0.9858	316.97 sec	0.9644	6525
Fashion-MNIST	0.9086	169.73 sec	0.8601	3775

2.1.4. Training results with different TH1 settings

With the same hyper-parameters settings shown in Table 2.1, different TH1 settings are applied to the model and the training results are shown in the Table 2.3 below.

Table 2.3: Training results with the different TH1 settings

Dataset	TH1	Training accuracy	Training time (sec)	Testing accuracy	Num of parameters
MNIST	**0.0005**	0.9872	266.39	**0.9677**	6600
	0.001	0.9872	266.46	**0.9677**	6600
	0.002	0.9872	264.84	**0.9677**	6600
	0.0035	0.9867	273.94	**0.9670**	6575
	0.005	0.9858	316.97	**0.9644**	6525
	0.0075	0.9848	257.66	**0.9627**	6250
Fashion-MNIST	**0.0001**	0.9127	146.52	**0.8598**	3975
	0.0005	0.9127	145.48	**0.8598**	3975
	0.001	0.9127	148.93	**0.8598**	3975
	0.002	0.9127	133.11	**0.8598**	3975
	0.003	0.9103	144.91	**0.8595**	3875
	0.005	0.9086	169.73	**0.8601**	3775
	0.0075	0.9020	103.53	**0.8533**	3425

To further analyze the correlations between the testing accuracies and the TH1 settings, curves between the testing accuracies and the TH1 settings are plotted as Figure 2.2 below.

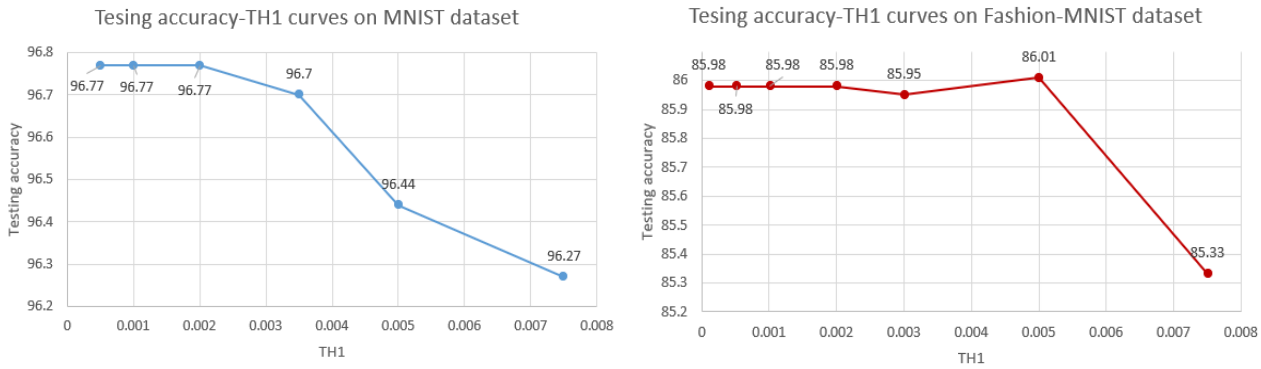


Figure 2.2: Testing accuracies-TH1 curves on MNIST dataset and Fashion-MNIST dataset

2.1.5. Discussions

From the experimental results above, the PixelHop++ obtains much higher training accuracies as well as the testing accuracies even with less training time, compared to the similar-scale BP-CNN models. This can definitely show that the PixelHop++ model is completely capable to handle the image classification tasks and it requires less training data than Deep Learning does.

From the different TH1 settings results, we can find that the testing accuracies are higher with lower TH1 generally. And the training time is also longer with lower TH1 values, because the model sizes grow larger with lower TH1. However, when TH1 is lower than a certain value (around 0.002), the testing accuracies will no longer grow higher. This might be a bottleneck for the TH1 setting and the model can be optimize with other hyper-parameter settings

2.2. Part (b) Comparison between PixelHop and PixelHop++

2.2.1. Abstract and motivations

In the previous section, the PixelHop++ model is built and applied to both the MNIST dataset and the Fashion-MNIST dataset. The PixelHop++ is actually an improved model based on the PixelHop model. The main improvement is that the PixelHop++ uses channel-wised Saab transform while the PixelHop model uses the ordinary Saab transform. In this part of problem, the performance of the PixelHop model and the PixelHop++ model will be compared and analyzed.

2.2.2. Approaches and procedures

The experiments settings for this part are similar to those of Section 2.1, but an additional model setting of PixelHop will be built and applied to both the MNIST dataset and the Fashion-MNIST dataset. The

PixelHop model uses three layers of ordinary Saab transform without any channel-wise processing and the other parts of these two models are exactly the same.

Also, the hyper-parameters are still following Table 2.1 with TH1 = 0.005 and TH2 = 0.001. The accuracies and the model sizes will be calculated to evaluate the model performance.

2.2.3. Experimental results

The training accuracies, testing accuracies and models sizes for both the PixelHop model and the PixelHop++ model are displayed in Table 2.4

Table 2.4: Training results for PixelHop and PixelHop++ on both MNIST and Fashion-MNIST

Dataset	Model	Training accuracy	Testing accuracy	Num of parameters
MNIST	PixelHop	0.9885	0.9650	133125
	PixelHop++	0.9858	0.9644	6525
Fashion-MNIST	PixelHop	0.9114	0.8617	74275
	PixelHop++	0.9086	0.8601	3775

2.2.4. Discussions

- **Comparisons of Training and testing accuracies of PixelHop and PixelHop++ models**

From the experimental results above, if using the TH1 and TH2 values shown in Table 2.1, PixelHop model can get a slightly higher accuracy for both the training dataset and the testing dataset. The superiority on accuracies for the PixelHop model can be a little higher if using the Fashion-MNIST dataset because the Fashion-MNIST dataset is more complex and harder to classify. The results is actually intuitive because both the MNIST dataset and the Fashion-MNIST dataset have only gray-scale images which can make the PixelHop++ ‘s “channel-wised” advantages useless. Therefore, the accuracies for these two models are very similar.

- **Comparisons of model sizes of PixelHop and PixelHop++ models**

If we make some further analysis on the model sizes for these two distinguished models, the results can be different. We can find that in the term of the number of features, the model size for PixelHop model is much larger than that for PixelHop++ model. Since the model size (or the feature dimensions) is so large for the PixelHop model, it will take much more time to train the model and also it will occupy much larger RAM spaces to run the training process. However, the accuracy results for these two models have little difference, so we can say that the PixelHop++ model is much more efficient compared to the PixelHop model.

2.3. Part (c) Error analysis

2.3.1. Abstract and motivations

In this part of problem, we will dig a little bit deeper to analyze the performance for the PixelHop++ model, by calculating the confusion matrixes for the testing datasets. Since a dataset often contains easy and hard classes, the confusion matrixes will indicate the classification ability of the model on different classes in the specific dataset.

2.3.2. Approaches and procedures

The training hyper-parameters are still the same as shown in Table 2.1, and the PixelHop++ model will be trained with the whole datasets of 60,000 images from the MNIST dataset and the Fashion-MNIST dataset, respectively. The confusion matrixes for these two datasets are calculated directly from the training samples and displayed as heatmaps. The analysis on the error rates for each dataset and the confusing pairs will be shown below.

2.3.3. Experimental results

The confusion matrix for PixelHop++ model trained with MNIST dataset is shown in Figure 2.3 below.

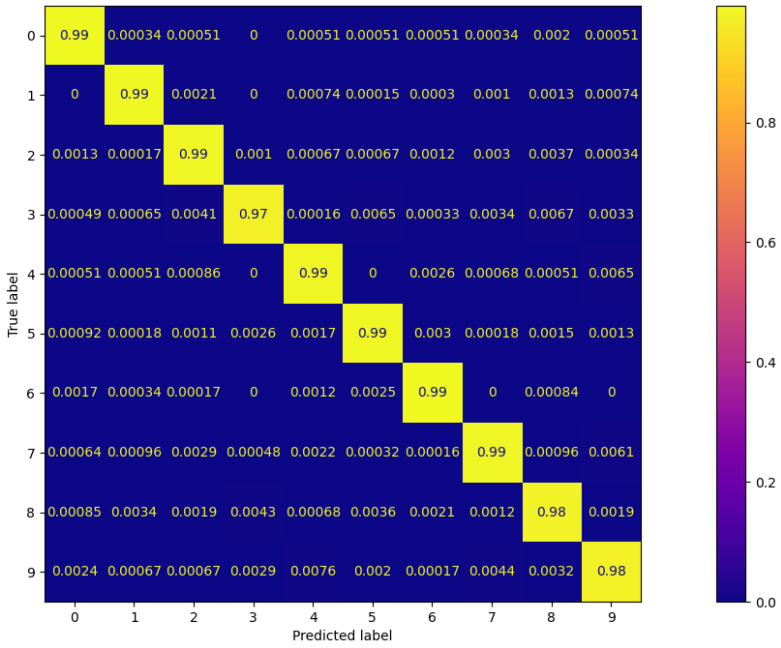


Figure 2.3 Confusion matrix for PixelHop++ model trained with MNIST dataset

The confusion matrix for PixelHop++ model trained with Fashion-MNIST dataset is shown in Figure 2.4 below.

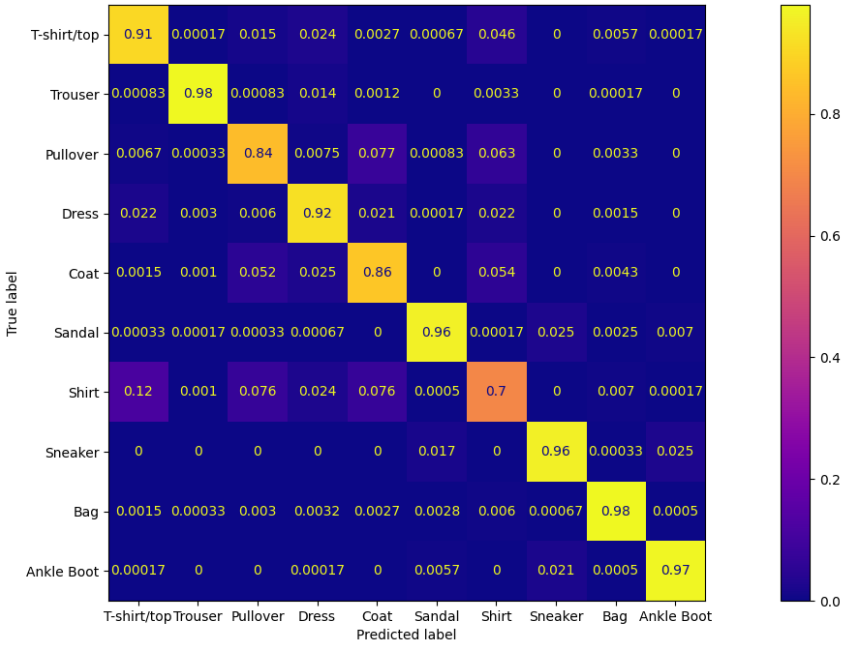


Figure 2.4 Confusion matrix for PixelHop++ model trained with Fashion-MNIST dataset

2.3.4. Discussions

1) For the **MNIST** dataset, nearly all the classes can reach an accuracy of over 99% except for the classes: *"3", "8" and "9"*. Among them , the class "3" yield the highest error rate of 3%.

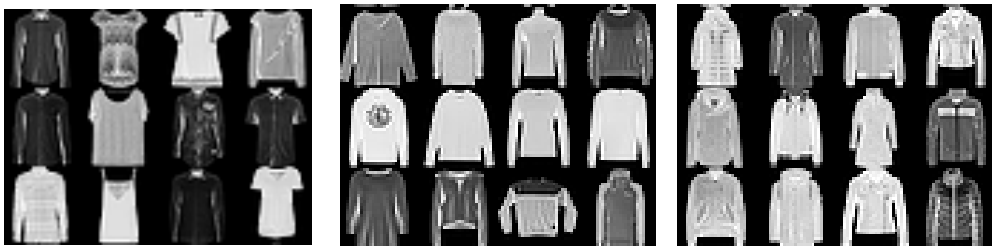
   For the **Fashion-MNIST** dataset, *"Bag"* and *"Trousers"* classes both yield the lowest error rates and class *"Shirt"* yields the highest error rate. These results are quite similar to the results of CNN models in Homework #5.

2) The top three error rate confusing class pairs in the **MNIST** dataset are: *"9" & "4"* (0.76%), "4" & "9" and "7" & "9" (0.61%). Two examples from these three classes are shown in Figure 2.5. We can find that the errors are very intuitive because the hand-written numbers from these three classes are very similar to each other and even human beings can make wrong judgements when confronting these three classes.

(a) pair of 7 and 9 (b) pair of 4 and 9

Figure 2.5 Examples of two confusing pairs from MNIST dataset

The top three error rate confusing class pairs in the **Fashion-MNIST** dataset are: “Pullover” & “Coat” (7.7%), “Shirt” & “Pullover” (7.6%) and “Shirt” & “Coat” (7.6%). Some example images from these confusing classes are shown in Figure 2.6. The errors from these classes are also very intuitive because the images from these three classes are very similar and they are quite hard to be classified.



(a) Shirts (b) Pullovers (c) Coats

Figure 3.4 Examples of three confusing classes from Fashion-MNIST dataset

3) To improve the performance of PixelHop++ model on these hard classes mentioned above, I’d like to propose two ideas. The first one is that we can combine the ideas of PixelHop++ and BP-CNN together and to form a new fusion model containing both the classic supervised learning and the weakly-supervised learning. That is, we can replace the XGboost classifier of the PixelHop++ model with some “supervised” or “Label-driven” classifiers, such as SVM and Random forests. We can still extract the features from the images using Saab transform or c/w Saab transform, but we input these feature vectors into SVM or random forests to train the classifiers and do the classification. This may be able to strengthen the ability of classification of the PixelHop++ model.

   Another idea I’d like to propose is that we can use some Data augmentation methods to augment the training for these confusing classes. For example, we could apply some image modification methods (rotating, flipping and adding noises) to the “Shirt”, “Coat” and “Pullover” classes’ images to expand the scale of training samples from these several classes and mix up the whole training dataset. I believe these techniques could do some help to improve the model’s performance on these confusing classes.