Name: Kuimu Ren
USC ID Number:1473482531
USC Email:kuimuren@usc.edu
Submission Date: 04/30/2024

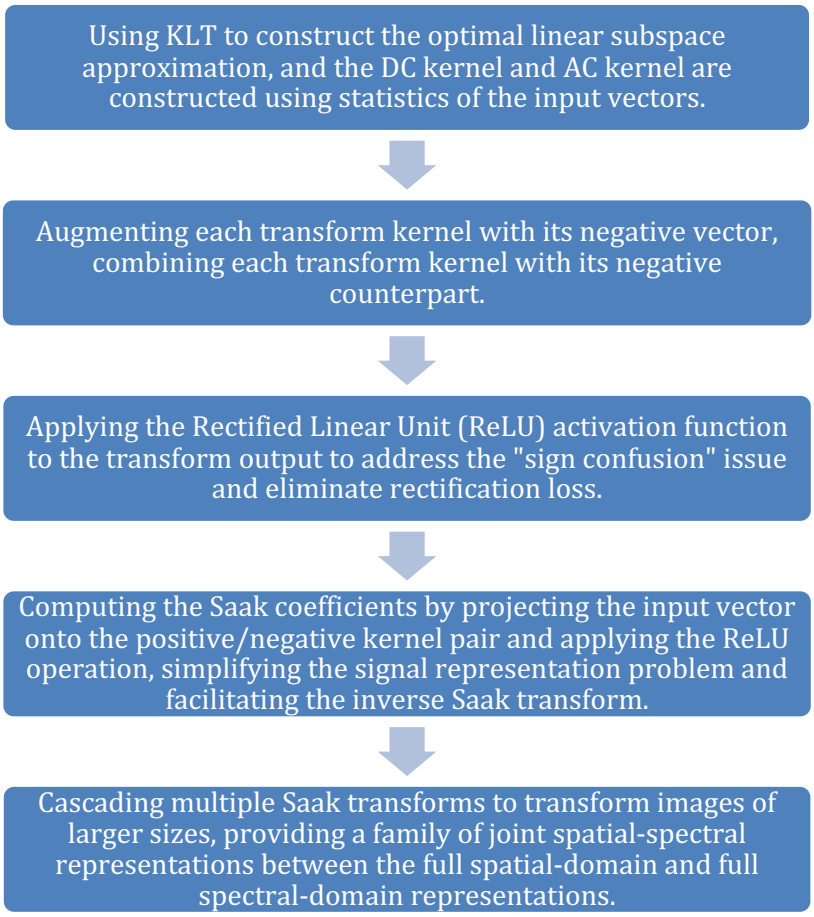
# Problem1

**Origin of Green Learning**

Larger, deeper models have emerged in recent years and continue to drive state-of-the-art (SOTA) results in various fields such as natural language processing (NLP) and computer vision (CV). However, despite the excellent results, it is important to note that the computations required for the SOTA model grow at an exponential rate. Not only does large-scale computing create a surprisingly large carbon footprint, but it also negatively impacts the inclusion of research and the deployment of real-world applications. Green deep learning is an increasingly hot research field that calls attention to energy use and carbon emissions during model training and inference. The goal is to produce new results with lightweight and efficient technology.

**(a) Feedforward-designed Convolutional Neural Networks (FF-CNNs)**
**Homework Answer and Discussion:**

1. The flow diagram of Saab transform:



2. The similarities between FF-CNN and BP-CNN:
   Both FF-CNN and BP-CNN utilize neural network architectures for processing input data. They consist of layers of neurons that perform computations on the input data to make predictions or classifications.
   Both designs incorporate convolutional layers for feature extraction from input data. These layers help in capturing hierarchical features from the input images, enabling the network to learn patterns and structures.
   Both FF-CNN and BP-CNN are commonly applied in image recognition tasks, such as object classification, segmentation, and tracking. They leverage the power of convolutional neural networks for processing visual data.

   The differences between FF-CNN and BP-CNN:
   The key difference lies in the optimization approach used to determine model parameters. BP-CNNs rely on backpropagation, an iterative optimization process that adjusts the network weights by propagating errors backward from the output layer to the input layer. In contrast, FF-CNNs do not use backpropagation for parameter optimization. Instead, they employ a data-centric approach, such as the Saab transform for convolutional layers and linear least squared regressors for fully-connected layers.
   BP-CNNs are end-to-end tightly coupled, where the model parameters at each layer are influenced by both input data and output labels. In contrast, FF-CNNs decouple the network into two modules: a data representation module (convolutional layers) and a decision module (fully-connected layers). This decoupling aligns with the traditional pattern recognition paradigm of separating feature extraction from classification/regression.
   Both BP-CNNs and FF-CNNs are vulnerable to adversarial attacks when the network model is fixed and known to attackers. However, FF-CNNs suggest adopting an ensemble method by fusing results obtained from multiple networks to enhance robustness. This approach is more feasible in FF-CNNs due to the ease of integrating multiple classifiers after feature extraction.

   The advantages of FF-CNN:
   FF-CNN design is mathematically transparent as it exploits data statistics to determine spatial-spectral transformations in convolutional layers without the need for data labels. This transparency enhances interpretability and understanding of the model.
   FF-CNN decouples the network into two explicit modules - data representation (feature extraction) and decision-making, aligning with traditional pattern recognition paradigms. This modular design simplifies the network structure and improves flexibility in integrating different techniques.
   FF-CNN allows sharing of convolutional layers across multiple tasks, reducing the need to design separate networks for each task. This facilitates the integration of various techniques and features, enhancing overall performance and efficiency.

   The limitations of FF-CNN:
   Like BP-CNN, FF-CNN is vulnerable to adversarial attacks, which can significantly degrade classification performance.

FF-CNN may require task-specific design for the fully connected layers, which connect the extracted feature space to decision labels. This customization may limit the generalizability of the model across diverse tasks without additional adjustments.

**(b) Understanding PixelHop and PixelHop++**
**Homework Answer and Discussion:**

1. Successive Subspace Learning (SSL) is a novel machine learning methodology that focuses on enhancing object recognition tasks, particularly in image classification. The SSL approach involves four key components:
   Successive Neighborhood Expansion: SSL expands from near to far neighborhoods in multiple stages. This means that it considers not only the immediate surroundings of a pixel but also progressively wider areas to capture more context and information.
   Unsupervised Dimension Reduction via Subspace Approximation: At each stage, SSL uses statistical correlations between attribute vectors associated with neighborhoods to find an approximate subspace. This helps in reducing the dimensionality of the data while retaining important information.
   Supervised Dimension Reduction via Label-Assisted Regression (LAG): SSL incorporates labeled data to further reduce the dimensions of the data. By leveraging regression techniques with labeled information, SSL can enhance the discriminative power of the model.
   Feature Concatenation and Decision Making: Finally, SSL concatenates the reduced features and uses them for decision-making tasks, such as image classification.

   Compare DL and SSL:
   DL is a parametric learning method that requires selecting a fixed network architecture to start with. It typically involves a large number of model parameters, leading to an over-parameterized network. SSL, on the other hand, adopts a non-parametric model. It focuses on unsupervised and supervised dimension reduction techniques to extract effective features without the need for a fixed network architecture.
   DL utilizes backpropagation (BP) for end-to-end optimization, which can demand significant computing resources during training, especially with deep networks. SSL follows a one-pass feedforward design, which results in lower training complexity compared to DL. The training process in SSL involves successive neighborhood expansion and dimension reduction.
   DL models are susceptible to adversarial attacks where small perturbations in input data can lead to incorrect predictions. SSL, with its use of techniques like PCA for dimension reduction, can filter out weak perturbations more effectively, making it challenging for attackers to conduct similar attacks.

2. Module 1 (PixelHop Unit):
   Module 1, also known as the PixelHop Unit, is responsible for unsupervised dimension reduction through the use of Saab filters. These filters are utilized to examine the near- and far-neighborhoods of selected pixels in the input data, which helps in extracting essential features from the input data.

   Module 2 (LAG Unit):

Module 2, referred to as the LAG Unit, is designed for supervised dimension reduction using regression matrices. This unit leverages label information to further reduce the dimensionality of the features obtained from earlier stages. The LAG Unit plays a crucial role in incorporating label-assisted regression techniques to refine the feature representation based on the specific classification task.

Module 3 (Classifier Parameters):
The Classifier Parameters module is responsible for integrating the reduced features from Modules 1 and 2 and using them to train a classifier for the final classification task.

3. Neighborhood Construction:
In the PixelHop unit, features are extracted successively from a pixel and its near-range, mid-range, and far-range neighborhoods in multiple stages. Similarly, in the PixelHop++ unit, the block diagram shows three PixelHop++ units in cascade, with Saab transform kernels of the same spatial dimension (5x5).

Subspace Approximation:
The PixelHop unit utilizes the Saab (Subspace Approximation with Adjusted Bias) transform for unsupervised dimension reduction. This transform helps control the rapid growth of output dimensions by decorrelating the covariance matrix into a diagonal matrix. In the PixelHop++ unit, a key modification is the introduction of the channel-wise (c/w) Saab transform. By decoupling a joint spatial-spectral input tensor into multiple spatial tensors and performing the Saab transform in a channel-wise manner, the model size is reduced while maintaining discriminant features for classification tasks.

The differences between the basic Saab transform and the channel-wise (c/w) Saab transform:
The basic Saab transform pools the input image from a grid of size $S_i \times S_i$ to a grid of size $S_{i+1} \times S_{i+1}$ through a max-pooling operation. It follows the design principle of using the same kernel and bias design for feature extraction and dimensionality reduction. The key difference with the channel-wise Saab transform is in the input tensor size. In PixelHop++, the input tensor is the spatial neighborhood size (e.g., $5 \times 5 = 25$), whereas in PixelHop, the input tensor is the multiplication of the spectral dimension and the spatial dimension ($25 \times K'$). The design of the channel-wise transform facilitates computation and allows for a smaller set of kernel parameters.
The basic Saab transform extracts features and performs dimensionality reduction on the entire image, while the channel-wise Saab transform extracts features from each channel image, reducing the size of the input tensor. The design of the channel-wise Saab transform is more suitable for the feature extraction requirements of PixelHop++, enabling efficient reduction of model size while preserving discriminative features required for classification tasks.

# Problem2

**PixelHop & PixelHop++ for Image Classification**

**Abstract and Motivation:**

PixelHop and PixelHop++ are image feature extraction methods based on hierarchical processing. They extract high-level features by processing pixel data in multiple layers. PixelHop++ is an improvement over PixelHop, introducing more techniques and optimizations to enhance feature extraction effectiveness and performance. These methods find wide applications in image processing and computer vision, particularly in tasks such as image classification and object detection.

**A) Building PixelHop++ Model**

**Approach and Procedures:**

In this part, we need to complete the three module methods in main：

First, we need to load the MNIST dataset and split it. Then preprocess the data, including data type conversion and normalization and select a balanced subset to ensure an equal number of samples for each class.

Second, on module1 we need to set the parameters for PixelHop and PixelHop++, including compression and shrinkage parameters. Then build the PixelHop++ model and train it and we can obtain features at different levels.

Third, on module2 we can obtain Hop3 features on the training set and then standardize the features.

Fourth, on module3, we need to initialize the XGBoost classifier and set the parameters. Then train the XGBoost classifier on Hop3 features. And use the trained model for prediction and calculate the training accuracy.

Last, print information such as model training time, model size, training accuracy and so on.

**Experimental Results:**

```
Training PixelHop++ model

==========================================>c/w Saab Train Hop 1
==========================================>c/w Saab Train Hop 2
==========================================>c/w Saab Train Hop 3
Done
Getting the number of K1 features...
Done.
Getting the number of K2 features...
Done.
Getting the number of K3 features...
Done.
The model size of PixelHop++ is: 6375
Running Module 2...
Getting the hop3 features for PixelHop++
(10000, 1, 1, 126)
(10000, 126)
training time: 353.3210504055023 seconds
Running Module 3...
Fitting xgboost on PixelHop++

Done
Getting accuracy of PixelHop++

The training accuracy is : 0.9768
```

Figure1: training time, train accuracy and model size of PixelHop++ on MNIST

```
Training PixelHop++ model

==========================================>c/w Saab Train Hop 1
==========================================>c/w Saab Train Hop 2
==========================================>c/w Saab Train Hop 3
Done
Getting the number of K1 features...
Done.
Getting the number of K2 features...
Done.
Getting the number of K3 features...
Done.
The model size of PixelHop++ is: 3875
Running Module 2...
Getting the hop3 features for PixelHop++
(10000, 1, 1, 74)
(10000, 74)
training time: 260.4845564365387 seconds
Running Module 3...
Fitting xgboost on PixelHop++

Done
Getting accuracy of PixelHop++

The training accuracy is : 0.8965
```

Figure2: training time, train accuracy and model size of PixelHop++ on FASHION_MNIST

```
Training PixelHop++ model

==========================================>c/w Saab Train Hop 1
==========================================>c/w Saab Train Hop 2
==========================================>c/w Saab Train Hop 3
Running Module 2...
Getting the hop3 features for PixelHop++
(10000, 1, 1, 128)
(10000, 1, 1, 128)
(10000, 128)
(10000, 128)
Running Module 3...
Fitting xgboost on PixelHop++

Getting accuracy of PixelHop++

The testing accuracy is : 0.9352
The training accuracy is : 0.9768
```

Figure3: train accuracy and test accuracy of PixelHop++ on MNIST

```
Training PixelHop++ model

==========================================>c/w Saab Train Hop 1
==========================================>c/w Saab Train Hop 2
==========================================>c/w Saab Train Hop 3
Running Module 2...
Getting the hop3 features for PixelHop++
(10000, 1, 1, 74)
(10000, 1, 1, 74)
(10000, 74)
(10000, 74)
Running Module 3...
Fitting xgboost on PixelHop++

Getting accuracy of PixelHop++

The testing accuracy is : 0.8147
The training accuracy is : 0.8957
```

Figure4: train accuracy and test accuracy of PixelHop++ on FASHION_MNIST

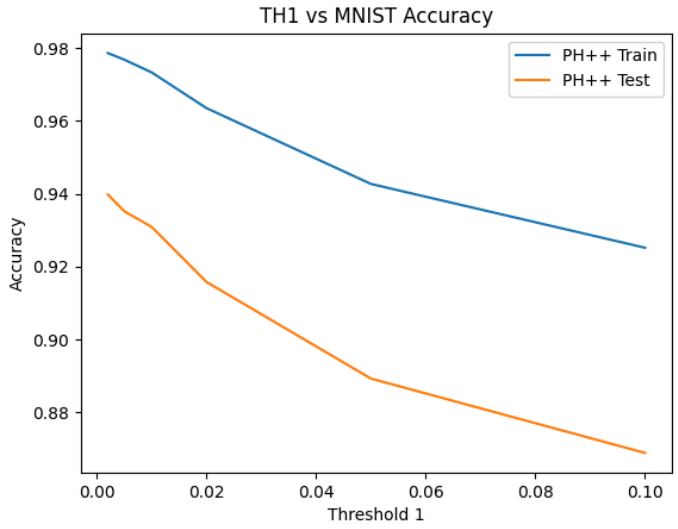



Figure5: the curve of TH1 vs. the test/train accuracy on MNIST

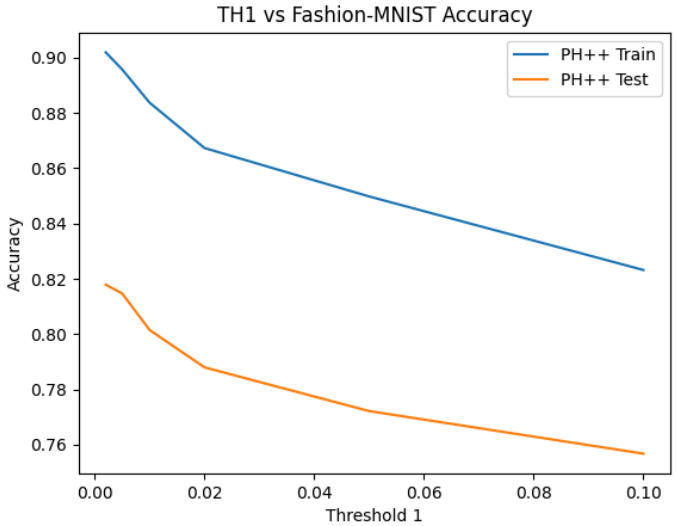



Figure6: the curve of TH1 vs. the test/train accuracy on FASHION_MNIST

**Homework Answer and Discussion:**

1. I use 10000 images for training. Figure1 show the training time, train accuracy and size of model size of PixelHop++ on MNIST. The training time is 353 seconds and train accuracy is 0.9768. And the total number of parameters of model is 6375.
   Figure2 show the training time, train accuracy and size of model size of PixelHop++ on FASHION_MNIST. The training time is 260 seconds and train accuracy is 0.8965. And the total number of parameters of model is 3875.

2. I apply model to 10000 testing images. Figure3 show the train accuracy and test accuracy of PixelHop++ on MNIST. The test accuracy is 0.9352.
   Figure3 show the train accuracy and test accuracy of PixelHop++ on FASHION_MNIST. The test accuracy is 0.8147.

3. The figure4 show the curve of TH1 vs. the test/train accuracy on MNIST and the figure5 show the curve of TH1 vs. the test/train accuracy on FASHION_MNIST.
   We can find that with the increase of TH1, the test accuracy decreases gradually. I think that as threshold1 increases, the model tends to retain features with higher energy and discard features with lower energy. This may result in the exclusion of some important but low-energy features, impacting the model's ability to represent the data and consequently reducing test accuracy. Therefore, as threshold1 increases, the model's generalization ability may be compromised, leading to a decrease in test accuracy.

**B) Comparison between PixelHop and PixelHop++**
**Experimental Results:**

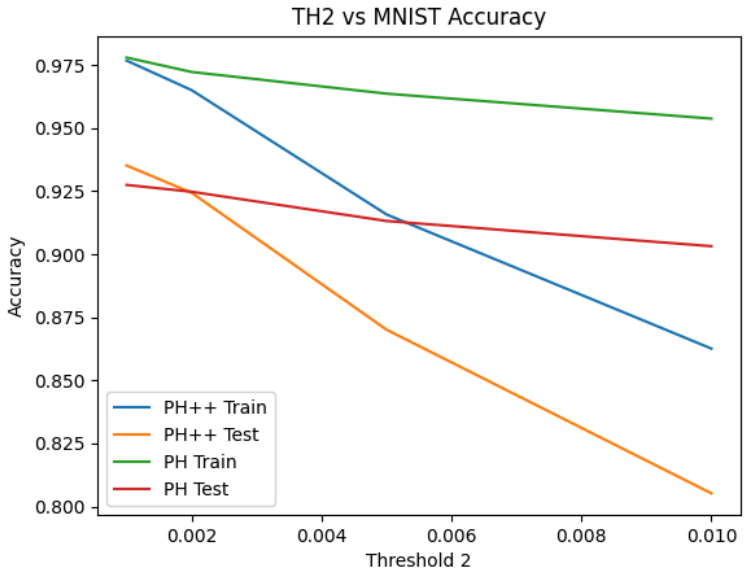



Figure1: the performance of PixelHop and PixelHop++ on MNIST

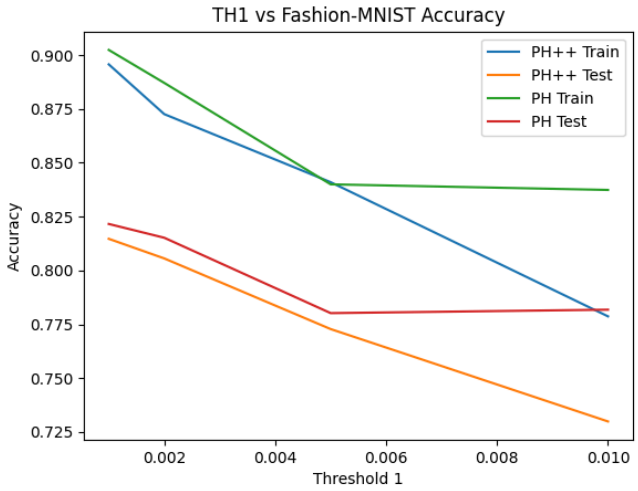



Figure2: the performance of PixelHop and PixelHop++ on FASHION_MNIST

```
Training PixelHop++ model

==============================================>c/w Saab Train Hop 1
==============================================>c/w Saab Train Hop 2
==============================================>c/w Saab Train Hop 3
Running Module 2...
Getting the hop3 features for PixelHop++
(10000, 1, 1, 128)
(10000, 1, 1, 128)
(10000, 128)
(10000, 128)
training time: 329.4285457134247 seconds
```

Figure3: the training time of PixelHop++ on MNIST

```
Training PixelHop++ model

==============================================>c/w Saab Train Hop 1
==============================================>c/w Saab Train Hop 2
==============================================>c/w Saab Train Hop 3
Getting the number of K1 features...
Done.
Getting the number of K2 features...
Done.
Getting the number of K3 features...
Done.
The model size of PixelHop++ is: 6425
```

Figure4: the model size of PixelHop++ on MNIST

```
Training PixelHop model

==============================================>c/w Saab Train Hop 1
==============================================>c/w Saab Train Hop 2
==============================================>c/w Saab Train Hop 3
Running Module 2...
Getting the hop3 features for PixelHop
(10000, 1, 1, 66)
(10000, 1, 1, 66)
(10000, 66)
(10000, 66)
training time: 74.13278460502625 seconds
```

Figure5: the training time of PixelHop on MNIST

```
Training PixelHop model

==============================================>c/w Saab Train Hop 1
==============================================>c/w Saab Train Hop 2
==============================================>c/w Saab Train Hop 3
Getting the number of K1 features...
Done.
Getting the number of K2 features...
Done.
Getting the number of K3 features...
Done.
The model size of PixelHop is: 122100
```

Figure6: the model size of PixelHop on MNIST

**Homework Answer and Discussion:**

1. Figure1show the performance of PixelHop and PixelHop++ on MNIST and figure2 show the performance of PixelHop and PixelHop++ on FASHION_MNIST.
   We can find that with the increase of TH2, although the accuracy of both methods decreases, PixelHop performs better than PixelHop++. Even though both methods exhibit a decline in accuracy as TH2 increases, the inherent feature selection strategy of PixelHop may be more robust or better suited for the given dataset, allowing it to outperform PixelHop++ in this scenario. On the other hand, in PixelHop++, the feature selection based on cross-entropy values may not be as effective in capturing the most relevant features under the influence of a higher TH2.

2. Figure3 and figure4 show the runtime and model size of PixelHop++. Figure5 and figure6 show the runtime and model size of PixelHop.
   PixelHop employ a more simplified or efficient feature selection mechanism, leading to lower computational workload during model training and inference, thereby reducing runtime but potentially resulting in a larger model size. On the other hand, PixelHop++ introduce more complex feature selection methods or processing steps, increasing algorithm complexity and computational load, which could lead to longer runtime and smaller model size compared to PixelHop.

**C) Error analysis**

**Experimental Results:**

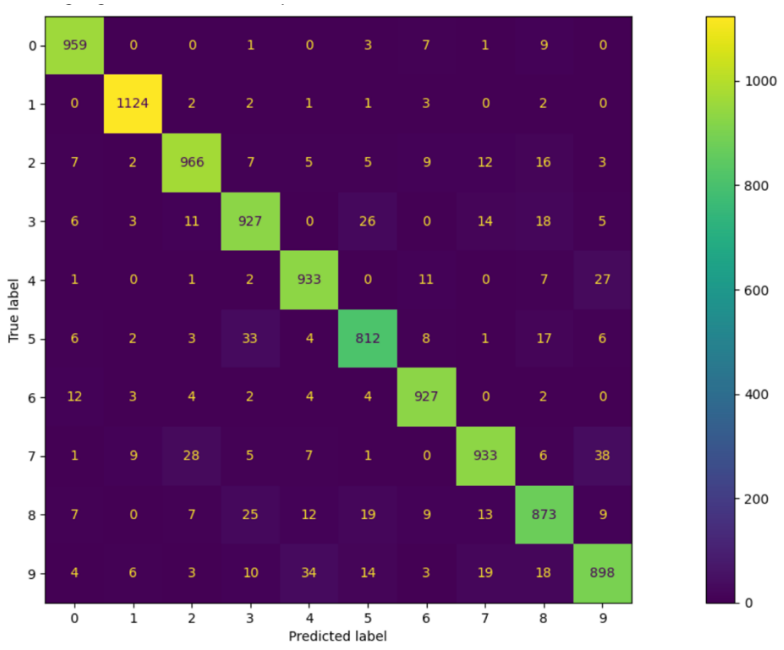



Figure1: the heat map of confusion matrix of PixelHop++ on MNIST

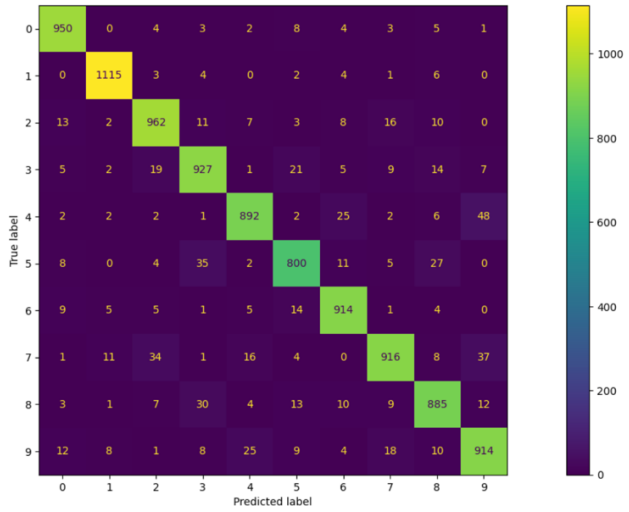



Figure2: the heat map of confusion matrix of PixelHop on MNIST

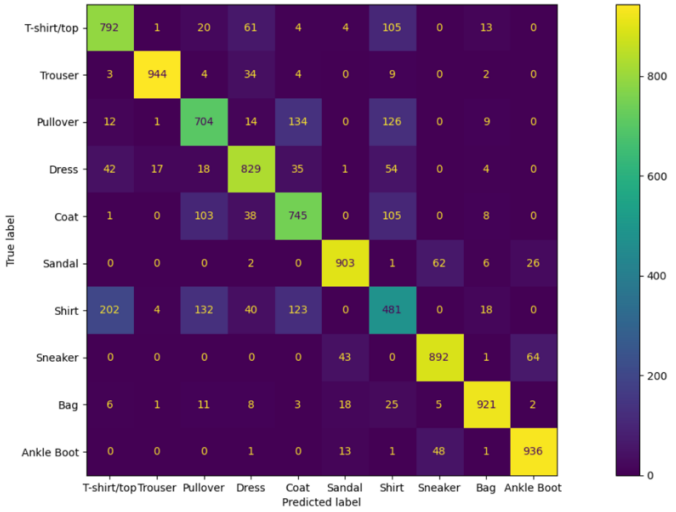



Figure3: the heat map of confusion matrix of PixelHop++ on FASHION_MNIST

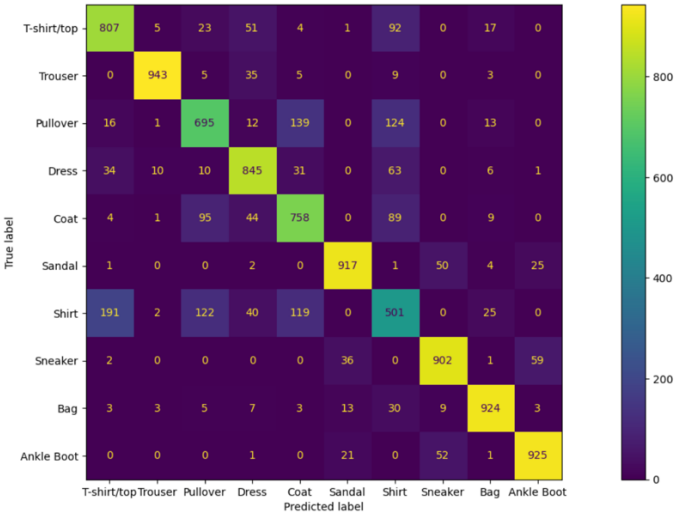


Figure4: the heat map of confusion matrix of PixelHop on FASHION_MNIST

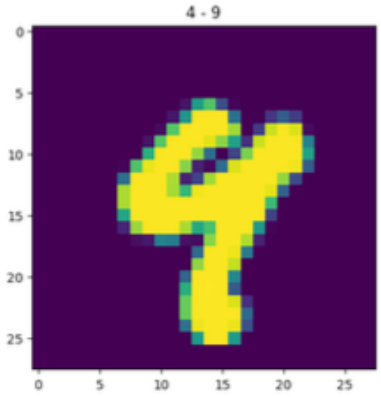



Figure5: confusion image (treat 4 as 9)

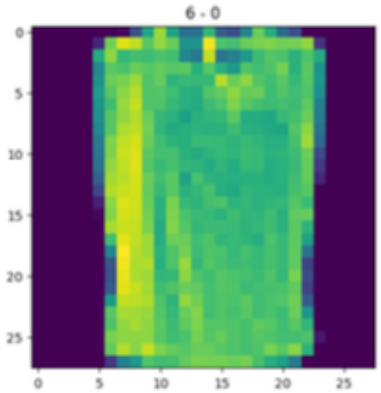



Figure6: confusion image (treat shirt as T-shirt)

**Homework Answer and Discussion:**

1. The figure from 1 to 4 show the heat map of confusion matrix of PixelHop++ and PixelHop on the MNIST and FASHION_MNIST.
   We can find in MNIST, object 5 get the highest error rate in PixelHop++ and object 1 get the lowest error rate in PixelHop++.
   And in MNIST, object 5 also get the highest error rate in PixelHop and object 1 also get the lowest error rate in PixelHop.
   In FASHION_MNIST, object ‘shirt’ get the highest error rate in PixelHop++ and object ‘Trouser’ get the lowest error rate in PixelHop++.
   In FASHION_MNIST, object ‘shirt’ also get the highest error rate in PixelHop and object ‘Trouser’ also get the lowest error rate in PixelHop.

2. We can find in MNIST, treat 7 as 9 get the highest error rate in PixelHop++ and treat 4 as 9 get the highest error rate in PixelHop. In FASHION_MNIST, treat ‘shirt’ as ‘T-shirt’ get the highest error rate in both methods.
   As we can see from figure5 to 6, these misjudged photos are only slightly different from the real labeled photos, and even humans can misjudge them. In addition, the image and texture features of the data set itself are also very fuzzy, and the image is likely to be

incorrectly classified under this kind of image which is partly similar to adversarial attack.

3. Augmenting the training data with techniques such as rotation, flipping, scaling, and adding noise can help expose the model to a wider variety of examples, making it more robust and improving its ability to classify difficult classes.
   Implement ensemble learning by combining multiple PixelHop++ models trained with different initializations or hyperparameters. Ensemble methods can help reduce overfitting and improve generalization, leading to better accuracy on challenging classes.