Problem 1: Origin of Green Learning (GL)

(a) (1)

As shown in the diagram, during fitting, Saab first removes DC feature from data by subtracting

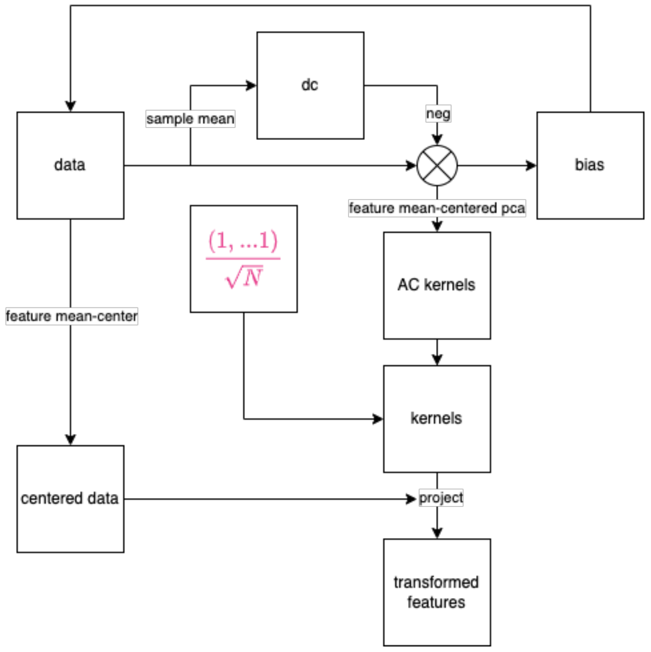

Figure 1: Saab flow chart

by the sample mean. Then it calculates PCA on the DC-corrected data and uses the eigenvectors as AC kernels. The DC kernel, essentially a normalized monotonous vector which returns the means of data projected onto it, is concatenated with AC features to form the complete kernel set. Data can then be projected onto the kernel set to get transformed features. Bias is a scalar value that's added to data element-wise to ensure that the projection will always return positive values. This removes non-linearity from the calculations and eliminates the need for activation function.

(2) Both rely heavily on feature selection, and both share a similar structure of procedurally transforming and downsampling data to arrive at a set of indicative features. However, the feature selection in BP-CNN is based on back-propagation, and therefore requires labels for its data to iteratively reduce error. The feature selection in FF-CNN is unsupervised and purely statistical manipulations on the data itself.

(b) (1) In SSL, data is divided into local neighborhood blocks which helps capturing spatial information. The blocks then undergoes Saab transform to get distinctive features whose energies are above a certain threshold. It should be noted that a subset of the training data is usually enough to get reasonably good features. The Saab-transformed features are also downsampled with max-pooling and fed into later layers where similar processes are repeated, potentially with different block sizes etc. The intention for having multiple layers is to allow SSL to capture increasingly more abstract information. The features from each layer can be either concatenated or used independently to serve as input features for supervised learning. SSL is quite different from DL. For starter, the feature selection process (module1 and module 2) is entirely feed-forward as discussed in (a), unlike DL which needs labels and back-propagation for feature selection, although both need labels to go from features to prediction. The notion of "nodes" in DL is substituted with kernels obtained from Saab transform, and there's hardly anything even remotely similar to DL weights.

(2) Module 1 constructs local neighborhood blocks, and fits kernels using a subset of the training data; it also transforms data using the kernels, energy-thresholds the obtained features, and passes features into the following layers. Module 2 Saab transforms the entire training data using the kernels obtained in Module 1 to get the features for supervised learning; these features are also energy-thresholded. Module 3 trains supervised model using the features as input.

(3) Neighborhood construction means dividing data into local windows. The channels from each location in the window are concatenated and the matrix containing all such concatenated windows is Saab-transformed to control the feature dimensions; this subsequent step is called subspace approximation. The entire matrix can be transformed in one large operation, which would be non-channelwise Saab transform. Alternatively, since AC features are PCA eigenvectors, they are orthogonal (independent) from each other and thus each channel can be transformed individually. This allows finer control of the model size. For example, if a feature already has low energy, we can expect its children in the next pixelhop unit to have even lower energy, and therefore we can either discard the feature all together if its energy is very low, or directly pass it as an output feature and not use it for later pixelhop units if it's energy is low, but not low enough to lose all significance. It should be noted that the very first pixelhop unit typically shouldn't use channel-wise transform, since its channels haven't been transformed and still tend to have correlations with one another; an example is the RGB channels in color image.

Problem 2: PixelHop & PixelHop++ for Image Classification

Motivation

Please see Problem 1.

Methods

The pixelhop framework was imported from this given repository. The codes were run on google colab with GPU accelerator enabled (thought it was really only useful for xgb training). Due to RAM constraints, module 1 was run on a balanced subset of training data containing 10000 samples. Module 2 was run on the entire training data, but in batches of 10000. The codes use pickle to save checkpoints. The rest is given in instructions.

Results and Discussion

(a) (1) The entire pipeline, including all 3 modules from pixelhop fitting to feature extraction to xgb training took about 25 minutes for MNIST and 30 minutes for Fashion-MNIST. The respective training accuracies were 0.99622 and 0.92917. The model sizes were 422975 and 151025. Model sizes were calculated by summing $prev\_K * win * win * K$ for all layers as given in the last discussion session.

(2) The respective testing accuracies were 0.9731 and 0.8623.

(3) The TH1 values I tried were [0.0001, 0.00056234, 0.00316228, 0.01778279, 0.1] for both MNIST and Fashion-MNIST. The corresponding testing accuracies were [0.9731, 0.9731, 0.9731, 0.957, 0.9154] for MNIST and [0.8623, 0.8623, 0.8614, 0.8399, 0.801] for Fashion-MNIST. The model sizes were [422975, 422975, 422975, 223850, 54650] for MNIST and [151025, 151025, 145125, 64575, 20075] for Fashion-MNIST. As TH1 increases, fewer nodes are passed into the following Hops for further splitting, so naturally the model size decreases, but this also discards information

so accuracy also decreases.

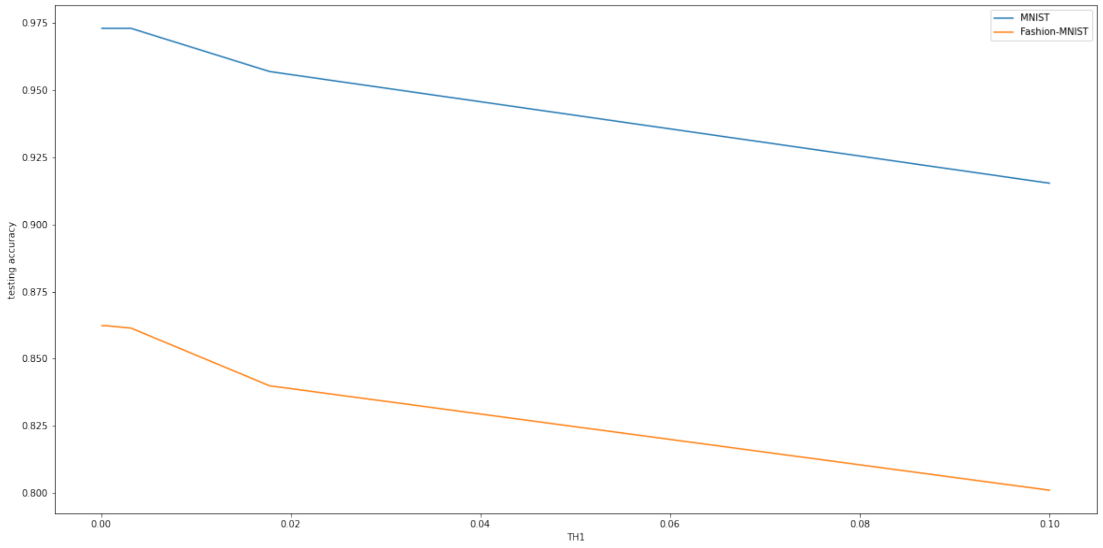

Figure 2: TH1 vs. the test accuracy

(b) (1) Figure 3 is plotted with TH2 values [0.008, 0.01196279, 0.01788854, 0.02674961, 0.04], while all other parameters, including TH1 are kept the same as in (a). TH1 doesn't play a role in non-channel-wise PixelHop, so it's not treated as a variable. For MNIST dataset, the test accuracies corresponding to the aforementioned TH2 values are [0.9516, 0.9442, 0.9335, 0.9108, 0.8639] for non-channel-wise PixelHop and [0.8891, 0.833, 0.6467, 0.3178, 0.3178] for channel-wise PixelHop++. For Fashion-MNIST dataset, the test accuracies are [0.8162, 0.8012, 0.7762, 0.7215, 0.7233] for PixelHop and [0.78, 0.7584, 0.7362, 0.6811, 0.6811] for PixelHop++. As shown in Figure 3, under similar parameters, PixelHop outperforms PixelHop++. This is understandable since Pixelhop++ essentially trades a little lost information for significantly smaller model size, as will be discussed in (2). PixelHop++ will not output as many features for the classifier to train on, so the performance suffers a bit.

(2) For MNIST dataset, the model sizes corresponding to the aforementioned TH2 values are [13775, 9200, 5875, 3525, 2225] for non-channel-wise PixelHop and [13275, 7000, 2575, 1150, 650] for channel-wise PixelHop++. For Fashion-MNIST dataset, the model sizes are [6225, 3525, 2225, 1450, 875] for PixelHop and [7425, 3750, 1775, 900, 600] for PixelHop++. PixelHop++ results in fewer trainable parameters than PixelHop because it stops low-energy features from undergoing further transformations in the following layers. These low-energy features would have produced even-lower-energy features anyways so little information is lost.

(c) (1) As shown in Figure 4, for MNIST dataset, the class with the highest error rate (i.e. lowest

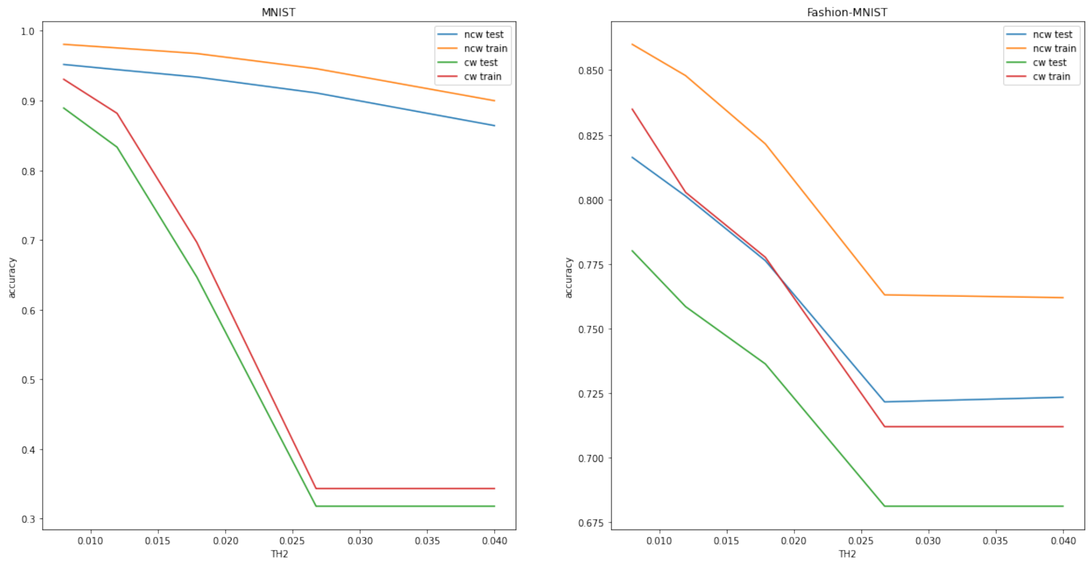

Figure 3: PixelHop++ vs PixelHop

accuracy) is 9 (0.048); the lowest error rate class is 1 (0.01).

$$\begin{bmatrix}
0.990 & 0.000 & 0.001 & 0.000 & 0.000 & 0.004 & 0.002 & 0.001 & 0.002 & 0.000\\
0.000 & 0.994 & 0.003 & 0.000 & 0.000 & 0.001 & 0.000 & 0.000 & 0.002 & 0.001\\
0.003 & 0.000 & 0.975 & 0.003 & 0.001 & 0.001 & 0.002 & 0.012 & 0.004 & 0.000\\
0.000 & 0.000 & 0.009 & 0.969 & 0.000 & 0.006 & 0.000 & 0.007 & 0.006 & 0.003\\
0.000 & 0.000 & 0.002 & 0.000 & 0.976 & 0.000 & 0.005 & 0.000 & 0.002 & 0.015\\
0.004 & 0.000 & 0.001 & 0.015 & 0.001 & 0.966 & 0.003 & 0.001 & 0.006 & 0.002\\
0.004 & 0.003 & 0.002 & 0.000 & 0.002 & 0.003 & 0.982 & 0.000 & 0.003 & 0.000\\
0.000 & 0.003 & 0.014 & 0.004 & 0.001 & 0.001 & 0.000 & 0.954 & 0.001 & 0.022\\
0.001 & 0.000 & 0.002 & 0.007 & 0.005 & 0.006 & 0.002 & 0.003 & 0.970 & 0.003\\
0.002 & 0.002 & 0.000 & 0.006 & 0.015 & 0.009 & 0.002 & 0.007 & 0.005 & 0.952
\end{bmatrix}$$

As shown in Figure 5, for Fashion-MNIST dataset, the class with the highest error rate is shirt (0.417); the lowest error rate class is bag (0.033).

$$\begin{bmatrix}
0.848 & 0.000 & 0.016 & 0.033 & 0.003 & 0.002 & 0.087 & 0.000 & 0.011 & 0.000\\
0.003 & 0.964 & 0.003 & 0.024 & 0.003 & 0.000 & 0.001 & 0.000 & 0.002 & 0.000\\
0.013 & 0.000 & 0.758 & 0.014 & 0.105 & 0.000 & 0.104 & 0.000 & 0.006 & 0.000\\
0.031 & 0.005 & 0.011 & 0.876 & 0.030 & 0.001 & 0.041 & 0.000 & 0.005 & 0.000\\
0.000 & 0.000 & 0.085 & 0.034 & 0.805 & 0.000 & 0.070 & 0.000 & 0.006 & 0.000\\
0.000 & 0.000 & 0.000 & 0.001 & 0.000 & 0.944 & 0.000 & 0.038 & 0.000 & 0.017\\
0.164 & 0.000 & 0.108 & 0.030 & 0.100 & 0.000 & 0.583 & 0.000 & 0.015 & 0.000\\
0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.033 & 0.000 & 0.928 & 0.000 & 0.039\\
0.003 & 0.000 & 0.005 & 0.005 & 0.002 & 0.006 & 0.008 & 0.003 & 0.967 & 0.001\\
0.001 & 0.000 & 0.000 & 0.000 & 0.000 & 0.011 & 0.000 & 0.038 & 0.000 & 0.950
\end{bmatrix}$$

(2) The top3 most frequent confused pairs, along with one example for each class, are shown below. Figure 6 shows that 7,4,9 in MNIST tend to be confused with one another. Figure 7 shows that T-shirt tends to be confused with shirt and pullover tends to be confused with coat. These are

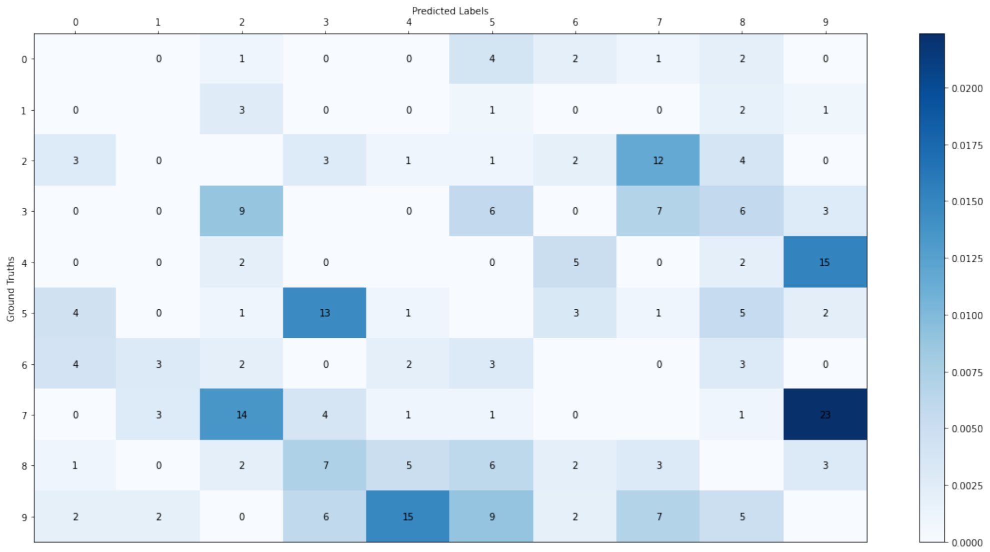

Figure 4: MNIST confusion

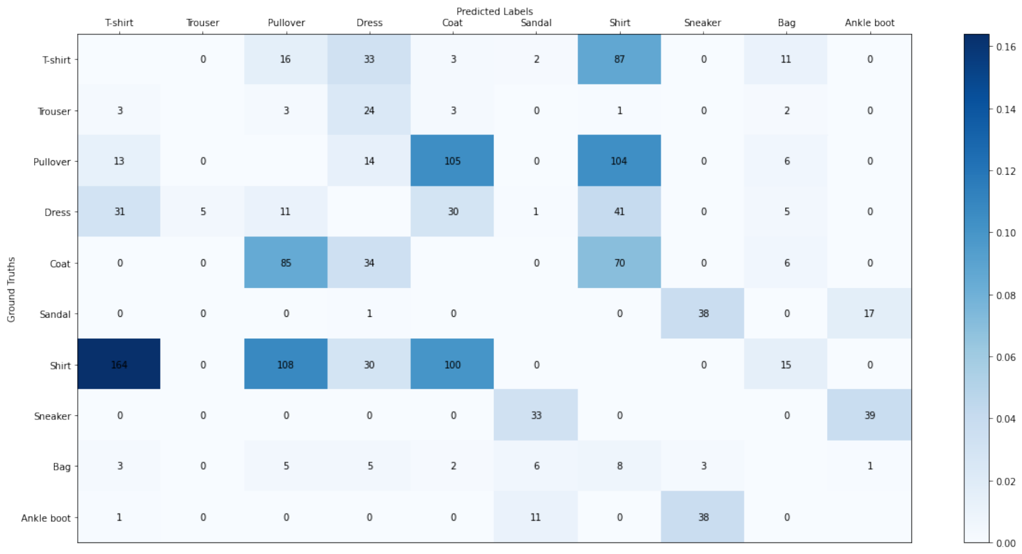

Figure 5: Fashion-MNIST confusion

sensible confusion pairs, and some of them can be hard even for human observer. For example, a 9 without a closed top looks very much like 4, and some of shirts can have short sleeves just like T-shirts.

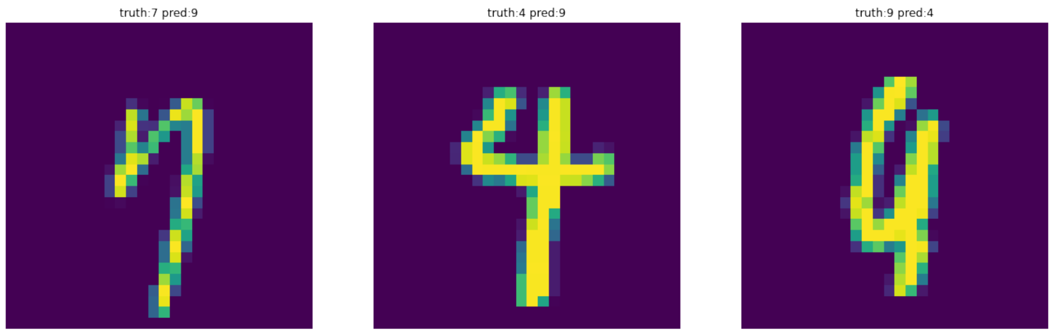

Figure 6: MNIST confused pairs

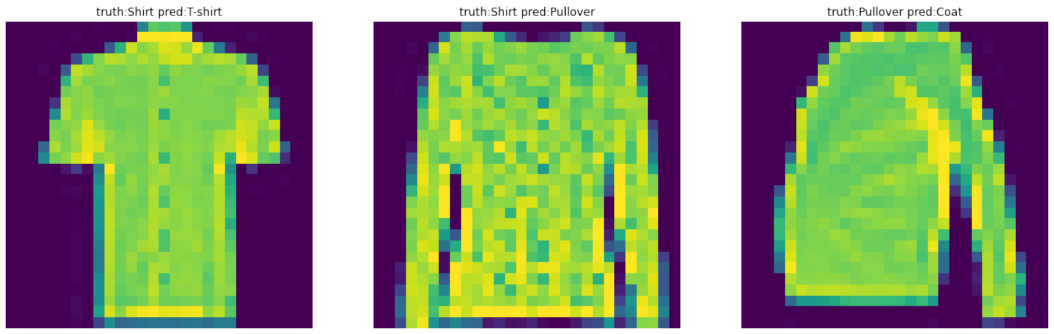

Figure 7: Fashion-MNIST confused pairs

(3) I think one improvement can be to do more training on the confused pairs. Get the PixelHop features that correspond to the confused pairs, and using only these features, continue training based on the baseline model.