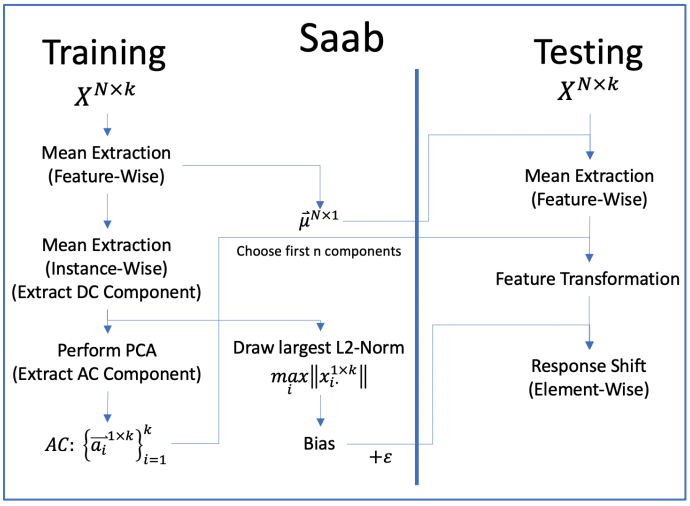
Homework 6 for 20 spring EE569
Successive Subspace Learning
Chengyao Wang
69615991816
chengyao@usc.edu

Problem 1, Understanding Successive Subspace Learning

1. **Feed-Forward Convolutional Neural Networks (FF-CNN):**
   FF-CNN are forwarded by[1] [1] mainly to address the problems of current deep-CNN, by mathematically intractable & computationally intensive being the biggest two. *Kuo et al.* are trying preserve the **"Feature Extraction – Decision Module"** structure of deep-CNNs while abandoning the backpropagation process. Every weight / parameter of FF-CNN is determined in a feed-forward manner, using the statistics of training dataset. Generally speaking, **"Feature Extraction"** module is theoretically supported by *PCA*, while the "**Decision module**" using *K-means* & *least squares regression*. They all come from traditional machine learning algorithms and thus are all mathematically grounded. Another important component of CNN [1] is avoiding the use of non-linear activation functions, specifically *ReLU* functions.

2. **Saab transform:**
   Saab transform plays the role of *feature extractor* in this FF-CNN framework and is designed to avoid the use of non-linearity by carefully designing the bias. Thus Saab got its name as "Adjusted bias". The procedure of Saab could be presented as follows:



   Notice that the number of AC kernels are *k-1*. This procedure is driven by the need to utilize unsupervised *PCA* method and avoid non-linear activations. Saab tried to remove the possibility of negative responses, thus needs a large enough bias to shift all the responses to positive. Since the overall strategy for determining "kernels" are *PCA* thus by restricting b_i to be the same, the bias vector naturally lands in the vector space spanned by $I = \frac{1}{\sqrt{K}}[1, 1, 1, \ldots\ldots, 1]$. We "pre-determined" one principle vector to be I, and by "pre-subtraction" of the DC component before PCA procedure, we could yield the rest un-correlated remaining principle vectors.

   Standard *PCA* procedure will generate *K* different eigenpairs, which generally are non-trivial solutions unless I happened to be one of the eigenvectors. However empirically, the least eigenvalue is close to zero. Since all kernels are of unit length, thus to make sure every response to be positive:

$$b = d\sqrt{k} \geq \max_{\vec{x}} \|\vec{x}\|$$

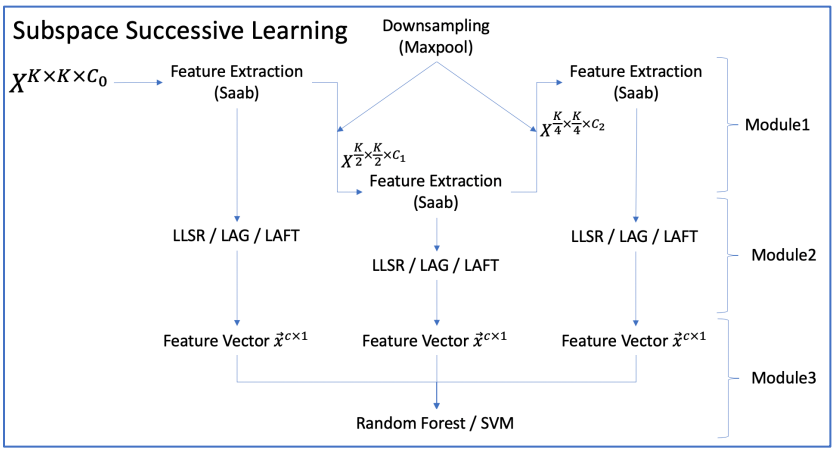
3. **Similarities between FF-CNN & BP-CNN:**
   The two are similar in many ways:
   a. They all have similar general structure, which can be separated into *feature extraction* part and *decision module* part.
   b. The feature extraction part (Saab & Conv) all have a sliding local window, or neighborhood approach to extracting feature vectors in numerous local scopes across the whole image. And also, correlation is both measured linearly.
   c. Decision module share the same computational graph structure - in a *"fully-connected"* way.

   But however, they are fundamentally different in many more aspects:
   a. Training BP-CNN involves gradient descent and backpropagation, but parameters in FF-CNN are all determined in one single forward pass.
   b. FF-CNN are designed to avoid the use of *ReLU*, which is considered as one of the sources of mathematical intractability.

[1] C-C Jay Kuo, Min Zhang, Siyang Li, Jiali Duan and Yueru Chen,"Interpretable convolutional neural networks via feedforward design," Journal of Visual Communication and Image Representation, vol. 60, pp. 346–359, 2019.

c. Though lacks theoretical proof, it's still widely accepted that labels in supervised problems play a crucial role and provides much information. However, **Saab & c/w Saab** is based on an unsupervised method, thus isn't utilizing the labels while BP-CNN do.

4. **Subspace Successive Learning (SSL) & Comparison to Deep Learning (CNN):**
SSL gets its name by the ***continuous feature extraction and dimensionality reduction stage***. "Subspace" has long been a topic in signal processing, where people seek approaches to represent high dimensional features with low dimension ones. "Successive" in *SSL* continuously distills and extracts information based on the ranking of eigenvalues of feature mapping (**Saab**) in different stages and further supervised dimensionality reduction of the feature vector using LLSR / LAG / LAFT etc. Each feature mapping beginning from the second stage could be taken as the "*subspace*" of the previous stage. Here is a high-level visualization of 3-stage SSL:



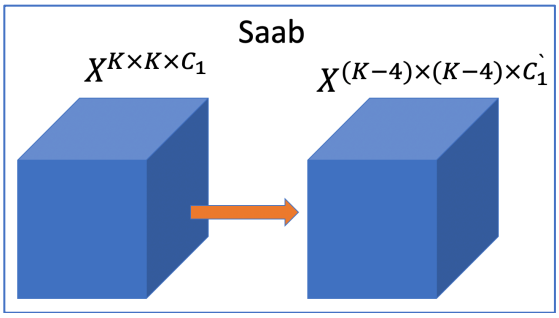
Note that the detailed size of X in each stage may vary because of other factors such as padding size, but the general idea is the same. Current concrete methods that adopts *SSL* include *PixelHop*[2], *PixelHop++*[3] etc. Since *FF-CNN* implicitly adopts *SSL*, thus *PixelHop* & *FF-CNN* actually shares much similarities and we are not repeating the comparisons between *SSL* & *BP-CNN* that are already mentioned above here, such as their different training methods (one-shot vs iterative). **But PixelHop / PixelHop++ implementation distinguishes their selves from the afore two by aggregated response vectors from all stages of down sampling, while standard CNN & FF-CNN only takes the highest-level response to the fc layers / decision modules.** Different *PixelHop* units works on different scales and generate response vectors in parallel, which is pretty much like model ensembles (*Bagging*). **Also, PixelHop / PixelHop++ introduces advanced classifiers from traditional machine learning methods, like SVM / RF to boost the model's predictive power.**

PixelHop & PixelHop++ share the same overall procedure despite some minor differences, mainly includes the introductions of **c/w Saab** & **cross-entropy based feature selection**. The reported accuracy on *CIFAR10* dataset is 66.81% for PixelHop++ (large) & 72.66% for PixelHop. The two methods are both scalable to different model sizes.

5. **Module in each Hop:**
   a. Module 1: unsupervised feature extraction & dimensionality reductions under different size receptive fields / neighborhoods. The former is done by *Saak / Saab / Channel-wise Saab* and the latter is controlled by eigenvalues / energy.
   b. Module 2: supervised dimensionality reduction that shares the same general form as *fc* layers. *Cross-entropy based feature selection* introduced in *PixelHop++* can be taken as one of its parts. Known techniques include *LLSR / LAG / LAFT*. **Note that module 2 actually possesses some predictive ability.**
   c. Module 3: traditional machine learning algorithms that performs the final step classification independently. It's used to enhance the overall predictive ability of the whole model and generate final prediction using the concatenated output of module 2 of different stages.
6. **Neighborhood Construction & Subspace Approximations:**
   a. Neighborhood construction serves as a preprocessing step before subspace approximation modules, e.g. *Saab*. It corresponds to *conv* operations in DL-CNN by also being conducted in a local scope manner. **For concrete methods, it's done simply by rearranging spatially neighboring feature vectors.** The window size is often 3-by-3 or 5-by-5. **The formulas shown below adopts 3-by-3 window while graph 5-by-5, and the notations are slightly different:**

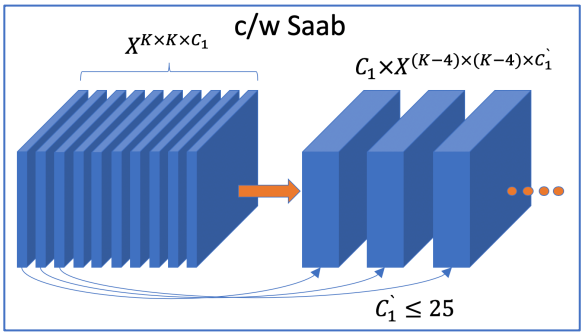
$$\begin{bmatrix} \overrightarrow{x}_{i,j}^{K\times 1} & \overrightarrow{x}_{i,j+1}^{K\times 1} & \overrightarrow{x}_{i,j+2}^{K\times 1} \\ \overrightarrow{x}_{i+1,j}^{K\times 1} & \overrightarrow{x}_{i+1,j+1}^{K\times 1} & \overrightarrow{x}_{i+1,j+2}^{K\times 1} \\ \overrightarrow{x}_{i+2,j}^{K\times 1} & \overrightarrow{x}_{i+2,j+1}^{K\times 1} & \overrightarrow{x}_{i+2,j+2}^{K\times 1} \end{bmatrix} \Rightarrow \left[\overrightarrow{x}_{i,j}^{K\times 1} \quad \dots \quad \overrightarrow{x}_{i+2,j+2}^{K\times 1}\right]^{9K\times 1} \underset{Reduction}{\overset{Saab}{\Rightarrow}} \overrightarrow{x}_{i+1,j+1}^{K\times 1}$$

[2] Yueru Chen and C-C Jay Kuo, "Pixelhop: A successive subspace learning (ssl) method for object recognition," Journal of Visual Communication and Image Representation, p. 102749, 2020.
[3] Yueru Chen, Mozhdeh Rouhsedaghat, Suya You, Raghuveer Rao, C.-C. Jay Kuo, "PixelHop++: A Small Successive-Subspace-Learning-Based (SSL-based) Model for Image Classification," https://arxiv.org/abs/2002.03141, 2020



PixelHop++ is not adopting padding method in this stage, as indicated by the dimensions presented. Subspace approximation methods are Saab / Channel-Size Saab in *PixelHop / PixelHop++* respectively. **The mechanics of Saab is already introduced above. *c/w Saab* is introduced [3] motivated by the observation that correlations spectrally is much weaker than spatial correlations by 1 order of magnitude, especially after the 1st Hop. Thus we could perform Saab channel wise, creating possibilities to decrease model size and time efficiency.** That is:

$$\begin{bmatrix} \overrightarrow{x_{i,j}}^{K\times 1} & \overrightarrow{x_{i,j+1}}^{K\times 1} & \overrightarrow{x_{i,j+2}}^{K\times 1} \\ \overrightarrow{x_{i+1,j}}^{K\times 1} & \overrightarrow{x_{i+1,j+1}}^{K\times 1} & \overrightarrow{x_{i+1,j+2}}^{K\times 1} \\ \overrightarrow{x_{i+2,j}}^{K\times 1} & \overrightarrow{x_{i+2,j+1}}^{K\times 1} & \overrightarrow{x_{i+2,j+2}}^{K\times 1} \end{bmatrix} \underset{=>}{do\ K\ times} K \times [x_{i,j} \quad \dots \quad x_{i+2,j+2}]^{9\times 1} \underset{Reduction}{\overset{Indepent\ Saab}{=>}} K \times \overrightarrow{x_{i+1,j+1}^`}^{N_k\times 1}$$



The differences between *Saab* & *c/w Saab* could be seen in the visualization above, but they are fundamentally identical.

b. Worth mentioning that *tree-representation of feature representation* method is one of the highlights of PixelHop++, but doesn't fit into any of the three modules. ***It's simply a visualization of ensemble of "valves" located after "max-pooling" module in each Hop++.*** *"Valves"* determine which response mapping shall go to the next *Hop++* while which are already suitable to feed into *module 2*, based on "*energy*" / *eigenvalues*. This method lowers the load in subsequent downstream *Hop* units and shrinks the model size.

7. Label-Assisted regression (LAG):

As an instance of module 2, the general purpose and functionality of LAG is already introduced. As an improved version of *LLSR*, its general structure is quite the same as *fc* layers in DL-CNN, with no activation functions & aren't trained using BP. ***LAG* inherits the idea of using pseudo-labels to construct a middle *fc* layer to avoid sharp dimension reduction from raw output to the number of classes.** *LAG* utilizes *K-means clustering* to construct *soft pseudo-labels* rather than using *hard labels* directly like *LLSR*. ***This mainly improves training stability of linear regression***. The procedure could be presented as:

$$Raw\ Output\ Vector \underset{fc1}{=>} \#_{Cluster} \times \#_{Classes}$$

a. For all the instances in the output of *module 2* that shares the same ground truth label, we perform *K-means clustering (K)* and therefore we get *K* cluster centroids vectors $\{c_i\}_{i=1}^{K}$.

b. To setup a *Least-Squares Regression (LSR)* problem, we need target vector for each instance. All instances no matter the label adopts the following structure:

$$\vec{y} = [\overrightarrow{y_0}^{1\times K}\ \overrightarrow{y_1}^{1\times K} \cdots \overrightarrow{y_{n-1}}^{1\times K}]^T$$

For an instance with label y_i, $\overrightarrow{y_j}^{1\times K}(i \neq j) = \vec{0}$. And:

$$\overrightarrow{y_j}^{1\times K} = [e^{-\alpha d(\overrightarrow{x_i},\overrightarrow{c_0})}, e^{-\alpha d(\overrightarrow{x_i},\overrightarrow{c_1})}, \cdots, e^{-\alpha d(\overrightarrow{x_i},\overrightarrow{c_K})}]^T$$

And we normalize by the sum of the elements. $d(\cdot)$ is Euclidean distance as the paper proposes, because it's the default choice in scenarios where we need metric functions.

c. With every $(\overrightarrow{x_i}, \overrightarrow{y_i}^{1\times NK})$ determined, we fit it into *LSR*. This is analytically solvable using pseudo-inverse.

Again, this process doesn't involve backpropagation and could be done in one forward pass, making it more time efficient than iterative gradient descent method in DL-fc.

Problem 2, CIFAR-10 Classification using SSL:

This section is an implementation of PixelHop++, we directly present the results & analysis without methodology description.

Some comments on implementation:

1. CIFAR10 dataset is downloaded from[4] and we've implemented a dataset loader class that supports fetching the dataset of certain size in *numpy.array* format. The arrangement of dimensions is consistent with the specifications of the modules provided: (N, H, W, C).
2. No dataset preprocessing is made except casting data type from *int* to *float32*.
3. We run the results on computing VM instance on GCP with 102GB memory to train the model under full training dataset size (50000). **Thus the size of the training set is consistent across all PixelHop++ units.**
4. We used the *RBF-SVM* to conduct the final classification, to avoid the tedious hyperparameter tuning process of *Random Forest Classifier*.

$$RBF(\vec{x}, \vec{y}) = e^{\frac{\|\vec{x}-\vec{y}\|^2}{\gamma}}$$

Notice that it could be proven that *RBF / Laplacian kernel machine* possesses infinite VC dimension and thus could achieve zero training error as long as there are no identical training points with conflicting labels. γ takes ${}^{1}/_{N_{features} \times var(X)}$, which is the default value of *sklearn.SVC* API[5]. ***We tried 5-fold cross-validation on γ using grid search, but the default value outperforms cross-validated values throughout a number of tests.***

5. Like assignment 5, we defined a *Recorder* class to record the configurations & running time information of the training & testing process. Its output is in JSON format.
6. Confusion matrix reported has *true label* on axis 0 and *predicted label* on axis 1. The elements are normalized by the total number of instances in each class.
7. Given the modules, the pipeline of training the *PixelHop++* system could be presented as:

$$PixelHop \xrightarrow{fit+transform} Maxpool \xrightarrow{Flatten} Sort\ Features\ CE \rightarrow FeatureSelection \rightarrow LAG \xrightarrow{fit+transform} SVM$$

The inference stage is also almost the same as training stage, where starting from $Maxpool$, the all stages are parallelizable. ***We turned on multi-threading for those stages (indicated by red font) by default***. The number of threads is equal to the number of stages we have. And thus, time consumption is reported on a per-stage basis. We used built-in library *threading*.

8. Model size calculation. Model size refer to the number of learnable parameters throughout the whole system, which includes:
    a. **c/w Saab Kernel**. For module *i*, input feature vector size is of $5 \times 5 \times C_i^{input}$, and we need $C_i^{output} + C_{i+1}^{input}$ number of them. With the first term referring to output of this current module and second being the size passed to the next module. Thus, $\mathbf{5 \times 5 \times C_i^{input} \times (C_i^{output} + C_{i+1}^{input})}$**.** The relationship between different modules would be as follows by tree-representation.

$$\mathbf{5 \times 5 \times C_i^{input} = C_i^{output} + C_{i+1}^{input} + \epsilon(TH2)}$$

$C_1^{input} = 3$ for CIFAR 10. **We could explicitly determine the three components in each *Hop* by printing out the dimensions before & after pruning.**

    b. **Feature Selection.** We need to save the cross-entropy based indices rankings for each module. Thus, $\boldsymbol{const = 1000}$. If this module is fed with less than $\boldsymbol{const}$ number of features, it's not counted.
    c. **LAG unit**. Least squares regression are linear transforms, thus $\left\|\boldsymbol{T_i^{M \times N}}\right\|_0 = \boldsymbol{M \times N}$.
    d. **Classifier.** Not counted as specified by the problem statement.

Notice that (a) is actually affected by *TH2* non-quantitatively, meaning that it's hard to find strict relationship between them & (a). We manually compute model size calculation by the dimension output during training.

Also worth pointing out, in this implementation of *PixelHop++*, every *c/w Saab* unit is outputting both leaf node & intermediate node. But as we can see from the dimension outputs, leaf nodes are not passed to the next stage.

At last, we're only reporting model size of 100% because models under weak supervision is found to be different from the former by a small number in output feature dimensions of PixelHop++ unit.

[4] https://www.cs.toronto.edu/~kriz/cifar.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Results:

Default Settings		Detailed Timetable (Train Set Size = 100%)			
Neighborhood Size	5-by-5		**Hop 1**	**Hop 2**	**Hop3**
Stride	1	***PixelHop++* Train**	0:32:40.23		
Max-pooling	2-by-2 (non-overlapping)	***PixelHop++* Transform**	0:03:40.02		
Energy Threshold 1	0.001	**MaxPool & Flatten**	0:00:21.30	0:00:15.19	0:00:00.00
Energy Threshold 2	0.0001	**Cross-Entropy Calculation & Sort**	0:29:43.67	0:25:34.64	0:02:04.08
Feature Selection (Cross-Entropy)	1000	**Feature Selection**	0:00:02.18	0:00:02.50	0:00:00.09
α in LAG units	10	**LAG Train**	0:00:31.96	0:00:39.26	0:00:31.07
Number of Centroids per Class	5	**LAG Transform**	0:00:00.21	0:00:00.24	0:00:00.15
Classifier	RBF-SVM	**SVM Train**	0:06:57.98		

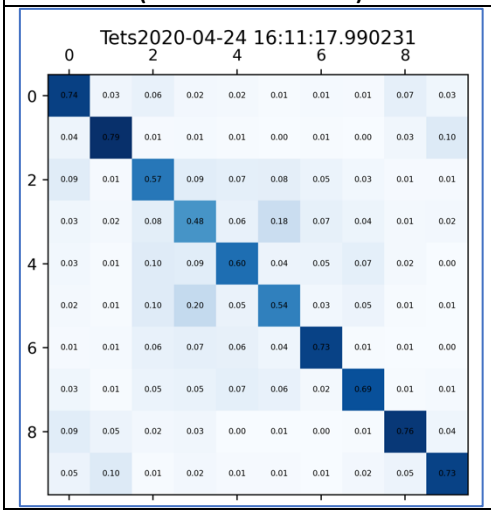
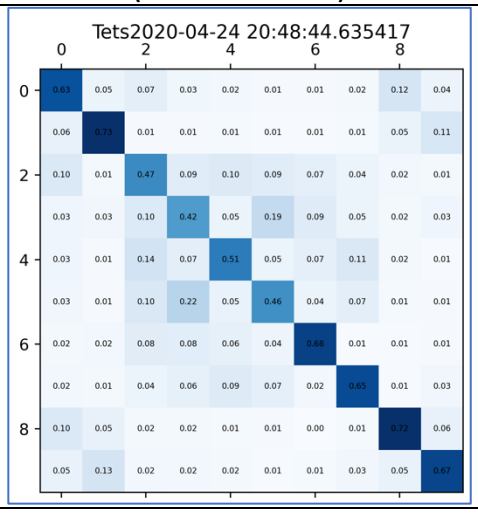
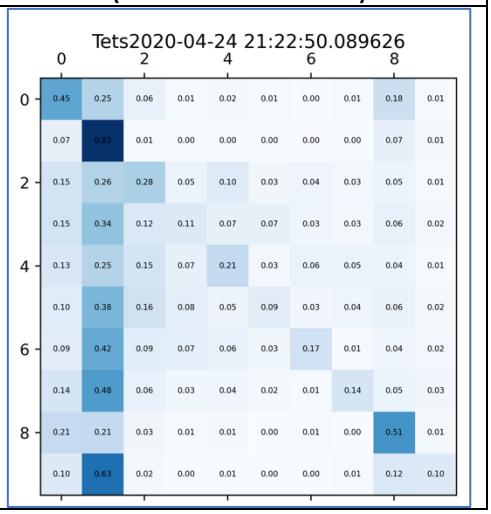
Summary under default settings

Training Set Size	**50000 (100%)**	**25000 (50%)**	**12500 (25%)**	**6250 (12.5%)**	**3120 (~6.25%)**	**1560 (~3.125%)**
Total Training Time	1:13:59.14	0:40:53.09	0:26:20.54	0:18:47.53	0:15:17.20	0:13:41.03
Train Accuracy	1.0	1.0	1.0	1.0	1.0	1.0
Test Accuracy	0.6613	0.6324	0.5938	0.5495	0.4699	0.288
Model Size (100%)	$\mathbf{25 \times (3 \times 42 + 25 \times 275 + 66 \times 529) + 1000 \times 2 + 1000 \times 50 \times 2 + 529 \times 5 = 1.15M}$					
Dataset Dimension (Train Set Size = 100%)		***PixelHop++***	**MaxPool & Flatten**	**Feature Selection**	***LAG***	**Concatenate**
	Stage 1	50000, 28, 28, 42	50000, 8232	50000, 1000	50000, 50	50000, 150
	Stage 2	50000, 10, 10, 275	50000, 6875	50000, 1000	50000, 50	
	Stage 3	50000, 1, 1, 529	50000, 529	50000, 529	50000, 50	

Note detailed dimension change between *Hops* (Leaf + Intermediate + Discarded):

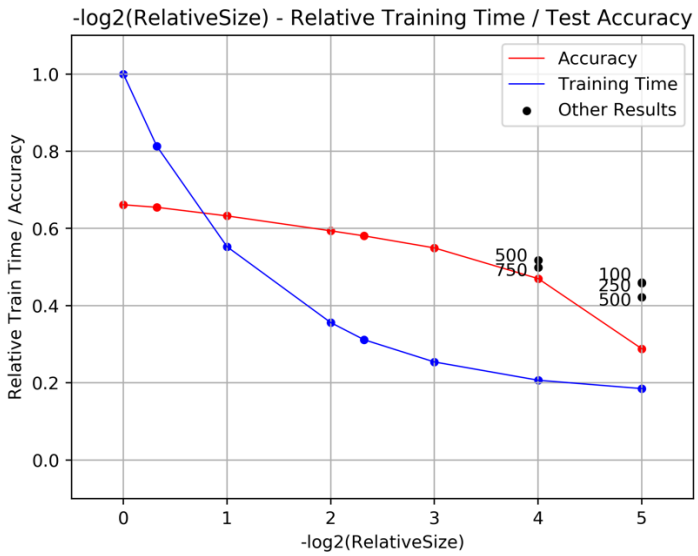
$$\boldsymbol{First\ Hop}: \mathbf{5 \times 5 \times 3 = 75 = 20 + 25 + 30}$$
$$\boldsymbol{Second\ Hop}: \mathbf{5 \times 5 \times 25 = 625 = 241 + 66 + 318}$$
$$\boldsymbol{Third\ Hop}: \mathbf{5 \times 5 \times 66 = 1650 = 526 + 0 + 112}$$

Confusion Matrix

(Train Set Size = 100%)	(Train Set Size = 25%)	(Train Set Size = ~3.125%)
		

Note that label encoding by [4] is: $airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck$

Other Trials

Train Set Size	Cross-Entropy Selected Features	Accuracy	Accuracy / Relative Training Time – Train Set Size (Default Setting, $\log_2 x$ scale)
40000 (80%)	1000	0.6547	
10000 (20%)	1000	0.5807	
3120 (~6.25%)	750	0.4989	
	500	0.5178	
1560 (~3.125%)	500	0.4219	
	250	0.4595	
	100	0.4579	

Comment & Analysis:

1. ***PixelHop++ system*** exploits strong general "feature extracting" ability, i.e. "learning" ability. The model could still maintain at a certain performance level under severe data insufficiency with training set $\approx {}^{1}/_{32}$ of the original size.
2. Tuning hyperparameters like the *number of cross-entropy selected features* we tuned above, or the *number of centroids in LAG units* are quite essential for a promising performance. **We got a ~20% accuracy boost by decreasing the number of afore mentioned former parameter.** The necessity of such process could be justified from a high-level. We are using cross-entropy to measure the information every dimension possesses, and data insufficiency would make a proportion of features that ranks the last useless, even negatively effects the downstream modules. Because we don't have enough training data to constraint the parameters and in this case, **the features themselves are "noises".** Considering the fact that ***RBF-SVM*** have infinite VC dimension, that leads to overfit.
3. **The training error is always 1.0 across all trials, as a result of kernel machine.** Normally, we should seriously consider overfit conditions to seek a better test accuracy. But in this case we've got performance reference from original paper, we just leave this issue.
4. **The shape of the two curves calls for better *training / time-accuracy* balance.** Accuracy drops almost-linearly while training time drops exponentially. But sadly, we care more about inference time rather than training time. It's still good news though, and on the other hand proves (1).
5. **Confusion matrix shows that *automobile* is the easiest class & *cat* is the hardest.** *We could see that the model is most confused between cats & dogs, quite obvious.* The reasons could be quite straightforward, *cat* & *dogs* are quite the same in terms of shape, body structure and maybe backgrounds since they are both pets. *The second pair would be automobiles & trucks*. This is also quite obvious, and the reason is quite similar to the first pair. If being stricter, we could see that the model implicitly splits the labels into:

   $$\{bird, cat, deer, dog, frog, horse\} \ \& \ \{airplane, automobile, ship, truck\}$$

   Surprisingly, the first group are all animals & latter some transport vehicles. ***In-groups misclassification rates generally are >0.03 while inter-groups misclassification rates are <0.03.*** We don't know if it's a coincidence, or there is some true underlying explanation.
   On top of the *groups* defined above, we could find a third ***inter-group*** pair which also confuses the model: *birds* & *airplanes*. Misclassification may be resulted from *flying birds' samples*, where we have a similar object & background.
6. Starting from the points above, we would propose the following ideas for performance improvement without heavily modifying the system structure & pipeline:
   1. Dynamic number of cluster centers in LAG. Determining suitable number of clusters has long been topic for *K-Means* clustering. One existing approach may be minimizing the ratio between inner-cluster variation & inter-cluster variation. We could use statistics to determine optimal K for every class in *FF* stage.
   2. Utilize statistics of feature-wise cross entropy values for feature selection. We already made simple trials and proven it matters.
   3. Add another cross-entropy based feature selection after LAG unit. Thought similar to (2), we further distill information and discard "noise" features.
   4. Change window size from 5-by-5 to 3-by-3, adjust *T1* / *T2* or LAG-α. Some normal parameter tuning.