HOMEWORK #6

Issued: 04/08/2020 Due: 04/26/2020

Problem 1: Understanding Successive Subspace Learning
Problem 2: Classification of CIFAR-10 using Successive Subspace Learning.
(Abstract and Motivation, Implementation, Results are written for both the problems in the same sections)

I. Abstract and Motivation:

In an effort to understand the underlying mathematics of Convolution Neural Networks, C C Jay Kuo proposed a mathematical model in the paper **"Understanding convolutional neural networks with a mathematical model"**[1]. In this work, a mathematical model called **RECOS transform** (Rectified Correlation in a Sphere) is introduced to explain the need of non-linear activation function between convolution layers.

Improvising on this concept while addressing the drawbacks of CNN such as vulnerability to adversarial attacks, lack of interpretability, and most importantly the complexity involved in training, Kuo et al. proposed a new way of representation of images through Saak transform and Saab transform. This opened the path to the development of Successive Subspace Learning via PixelHop method and other improved versions of it. In the subsequent sections, we discuss each topic in detail.

RECOS Transform: This gives a plausible explanation for the need of non-linear activation function in Convolutional Neural Networks (CNN). It posits that the non linear activation is needed to avoid sign confusion problems that occur in the subsequent convolutional layers. The result is using ReLu to clip the negative correlations between the input and the filter and retain just the positive correlations. This can be smartly interpreted as navigating the sign confusion problem.

Figure 1 shows the Sign confusion problem and Rectified Correlation sphere representation.

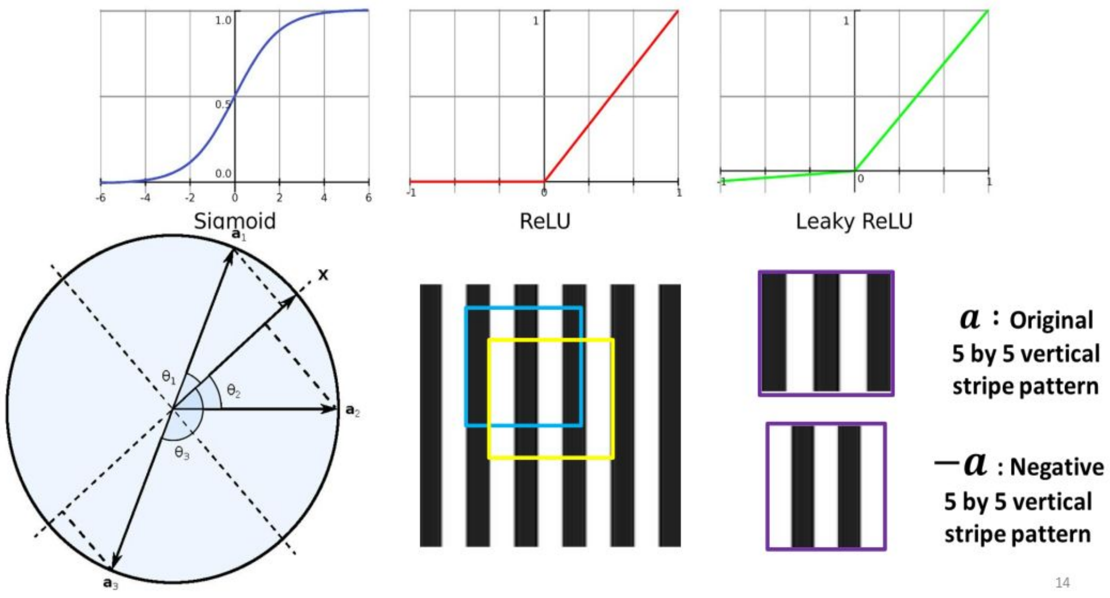

Sigmoid
ReLU
Leaky ReLU
a : Original 5 by 5 vertical stripe pattern
$-a$: Negative 5 by 5 vertical stripe pattern

Figure 1: Sign confusion problem and Rectified Correlation sphere representation.

Consider, the input f and the anchor vectors a_k. Then the projection of the input vectors on all the anchor vectors in given by,

$$p_k = a_k^T f, \text{ k = 0,1,.....K}$$

If we clip off the negative correlation, then the rectified response is given by,

$$g_k = \{p_k \text{ if } p_k > 0, \ 0 \text{ if } p_k \le 0\}$$

Inverse RECOS transform: We perform inverse RECOS transform to reconstruct the input signal back from the projected values. But there will be an approximation error while reconstruction the signal. This is given by,

$$f = f' = \sum_{k=0}^{K} \alpha_k a_k$$

From the above equation above, we can say that the approximation of input signal is the linear combination of the anchor vectors. So, the projection is given by,

$$p_k = a_k{}^T f' = a_k{}^T \sum_{k'=0}^{K} \alpha_{k'} a_{k'}$$

So based on this foundation, Saak transform, Saab transform, PixelHop and PixelHop++ has been developed. All these will be explained in detail in the subsequent sections. Also, we train the PixelHop++ model on CIFAR10 dataset and analyse the results.

II. Approach and Implementation:

1. The github code given is cloned and added in the **Google Colab platform** (Github Code: https://github.com/USC-MCL/EE569_2020Spring).
2. The implementation is done in Google Colab (a free online platform with GPU access) using Pytorch, Scikit-learn and other peripheral libraries.
3. The CIFAR-10 dataset is loaded using Pytorch and normalizer (wrapping the values in between 0-1).
4. Transform the training set to a new dimension which is of the form **(num_train_data, spatial dim, spatial dim, num_channels).**
5. Randomly sample 10,000 images from the training data in a balanced way i.e., each class should get an almost equal number of images.
6. Define Shrink args, Concat Args and Saab args as suggested in the hyper-parameter table.
7. Write callback function *shrink* based on the requirement.
8. *Max pooling* is implemented inside *shrink* function with a pooling size of 2*2.
9. Train PixelHop unit using randomly selected 10000 images and save the model for future.
10. Use the transform function on all 50,000 training images and save the result of module 1 output of all three pixelhop units.
11. Reshape the output of module 1 to (50000 * features).
12. Use cross entropy to select top 50% features with lowest cross entropies.
13. Use LAG unit to train the regression matrix using (50000 * newFeatures).
14. Concatenate the output of all the LAG units and use it as input to the classifier.
15. Random Forest classifiers are used and hyper-parameters are tuned for better accuracy.
16. **Error Analysis** is performed using a confusion matrix, heat-map.
17. **Weak supervision:** We reduce the number of training images to ¼, ½, ⅛ after module 1 and analyse the accuracy on test images.

18. Report the final accuracy, model size and chosen hyperparameters.
19. Analyse the reasons for errors in particular classes and propose ideas to improve the result.

III. Results and Conclusion:

1) Feedforward Convolutional Neural Networks (Question and Answers):

Deep learning has changed the paradigm of computer vision with it’s stellar accuracy on most of the vision tasks. The end-to-end architecture using back propagation to optimize a cost function proved to be efficient. The downside of such deep learning approaches include the requirement of huge training dataset, hardware resource and takes huge amounts of time for training. Deep learning approaches are often considered as a black box and a brute force method. The interpretability of math is a daunting task because of non-convex optimization.

Considering all these downsides, a new approach which is more transparent, more reliable and easier to interpret was required. Kuo et.al proposed a data centric method which uses the local statistics of the image to concisely represent it as a feature vector. Kuo et.al designed a Feedforward Convolution Neural Network where the model parameters are obtained using PCA instead of optimizing a cost function using back-propagation. The model parameters in this method are obtained using Saak transform [1] (Subspace approximation via augmented kernels) and Saab transform [2] (Subspace approximation via adjusted bias). We discuss Saab transform in detail in the following section.

Saab (Subspace approximation via adjusted bias) Transform: The entire idea of Saab transform is representing the image in a transformed feature space with reduced number dimensions. We can understand this in the viewpoint of Linear Algebra. The image in the input space, the intermediate results and also the class labels in the output space can be treated as vectors lying in a vector space. In each layer, we do transformation from one vector space to the other.

The same Philosophy holds for CNNs as well. The architecture of CNN connects the input Image space to the decision space. The key difference between CNN and Saab is that CNN learns the match filter to extract the features whereas Saab transform uses PCA to extract discriminant features from the image. While both the methods reduce the dimension of the input image before reaching the fully connected layers.

The transformation is done using two fundamental ideas: a) Dimension Reduction using Principal Component Analysis (projection), b) Training the extracted samples to determine the object classes. Dimension reduction process is an unsupervised learning whereas training the samples will be supervised learning. By employing this idea, we can clearly separate Convolution layers from fully connected layers. So, in Successive Subspace Learning the architecture will not be end-end tightly coupled.

Design of convolutional layers in a feedforward way: It is important to understand the role of computational neurons and non-linear activation in CNNs. Consider an N dimension input vector $x = (x_o + x_1 + ... + x_{N-1})^T$, an N dimension learnable filter weights $a_k = (a_0 + a_1 + ... + a_{N-1})^T$ and a bias b . The affine operation is given as follows,

$$y = \sum_{n=0}^{N-1} a_n x_n + b$$

The affine operation is followed by a non linear activation which is given by,

$$z = \phi(y) = max(y, 0)$$

In CNN the filter weights are determined using back propagation. But in SSL, the filters or **anchor vectors** are considered as a **set of vectors which spans a linear subspace.** If bias is set to zero i.e., $b = 0$ then,

$$y = a^T x .$$

The output vector is interpreted as the projection of input vector x onto the subspace spanned by anchor vectors a_k .

Finding Anchor Vectors: This approach derives the anchor vectors from the statistics of the image. Consider the patch of $N * N$ and reduce which corresponds to a vector of dimension N^2 . We combine such samples all across the image to form a covariance matrix. Using the covariance matrix we get eigenvectors corresponding to the maximum eigenvalues which form the anchor vectors for that layer (K anchor vectors). The next stage input is obtained by projecting the input patches of $N * N$ to each of the anchor vectors to obtain K spectral components. Figure 1 shows the projection of input vectors onto the anchor vectors.

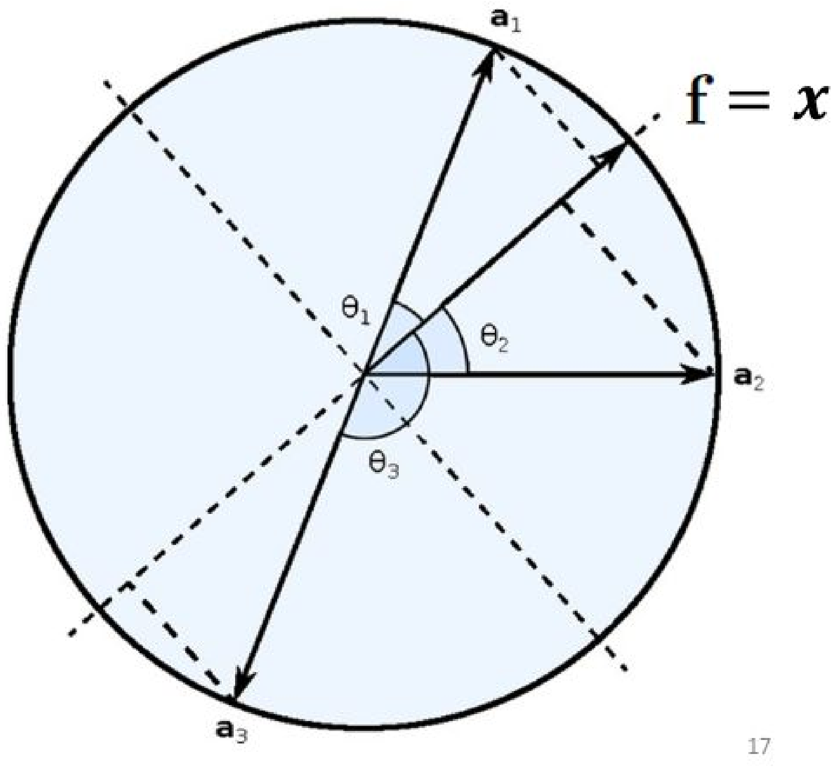


Figure 1: Projection of input vectors onto the anchor vectors.

Sign confusion and non-linear activation: In CNNs the non linear activation is used in between the convolution layers to avoid the sign confusion problem. When there is a negative correlation between the filter and the input and if the filter weight of the next layer is also negative, the convolution layer can't differentiate between the negated patterns. This problem is uniquely handled in Saab transform with using any activation function. Figure 2 shows the sign confusion problem occurs in CNNs. The name Subspace approximation via adjusted bias is mainly because of the way it handles sign confusion problems using bias variables.

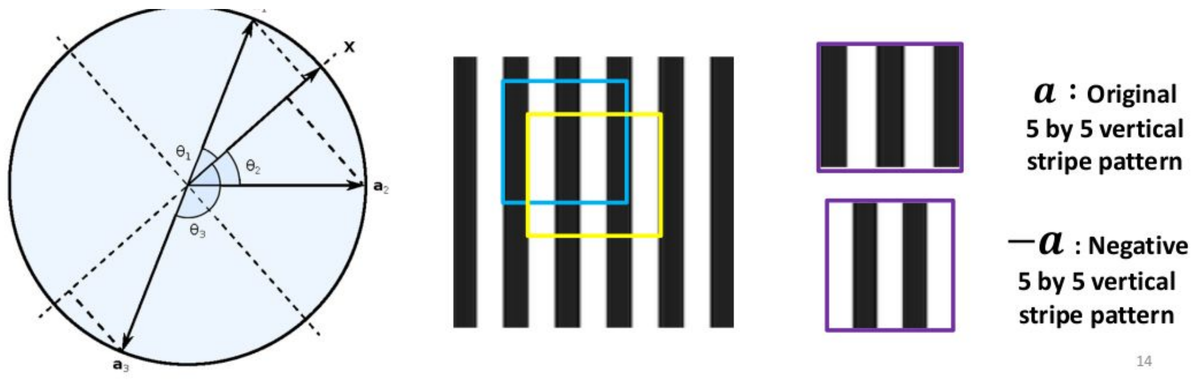


Figure 2: Illustration of Sign Confusion Problem.

Spatial Spectral Filtering: To appreciate the elegance of Saab transform we must understand the difference between **image representation** and **image features.** We can obtain different representations of images by transforms (Fourier transform, Discrete Cosine transform, wavelet transforms etc). The image features are obtained extracting edges, contours and salient point detection. We can draw a clear line between CNNs and Saab transforms based on these two concepts. The process of input vector space transformation by extracting anchor vectors is explained in the previous section. The main purpose of Spatial Spectral Filtering is to **increase the discriminant power of some dimensions.**

Anchor vectors and Bias Selection: The main goal transform is finding anchor vectors and bias term. To find the anchor vectors, we set the bias $b = 0$ and divide the anchor vectors into **AC and DC anchor vectors**. The entire subspace is represented as,

$$S = S_{AC} + S_{DC}$$

S_{DC} contains $a_0 = \frac{1}{\sqrt{N}}(1, 1, ...1)^T$ i.e., DC anchor vector a_0 spans the DC subspace. Accordingly, S_{AC} contains $a_k,\ k = 1, 2...., K-1$ i.e, AC anchor vectors a_k spans the AC subspace.

For an input vector $x \in R^N$, we can project the x to DC subspace to obtain,

$$x_{DC} = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x_n$$

AC and DC subspaces are orthogonal complements. So, AC component of the input is given by,

$$x_{AC} = x - x_{DC}$$

Bias selection is done on two constraints,

1. Bias terms should make the affine transformation always positive so that the non-linear activation automatically holds.

$$y = (\sum_{n=0}^{N-1} a_n x_n + b) \geq 0$$

2. Equal bias terms.

$$b_0 = b_1 = b_2 = ...b_{K-1}$$

Thumb rule to select bias term to satisfy the above constraints is,

$$b_k \geq max\ ||x||,\ k = 0, 1, ...K-1$$

Spatial Pooling: In CNN, pooling operation is done to reduce the spatial size which helps to remove redundancy and also to reduce the storage space. In Saab transform, convolution and pooling operations are combined. This is done in 2 steps,

1. Consider a $6 * 6$ patch in the input sample, we first project the the patch to smaller patch size $5 * 5$.
2. Project the $5 * 5$ patch to all the anchor vectors.

In this way, we reduce the spatial dimension by preserving the spectral dimension.

Figure 3 shows the FeedForward design of the Convolutional layer using subspace approximation.

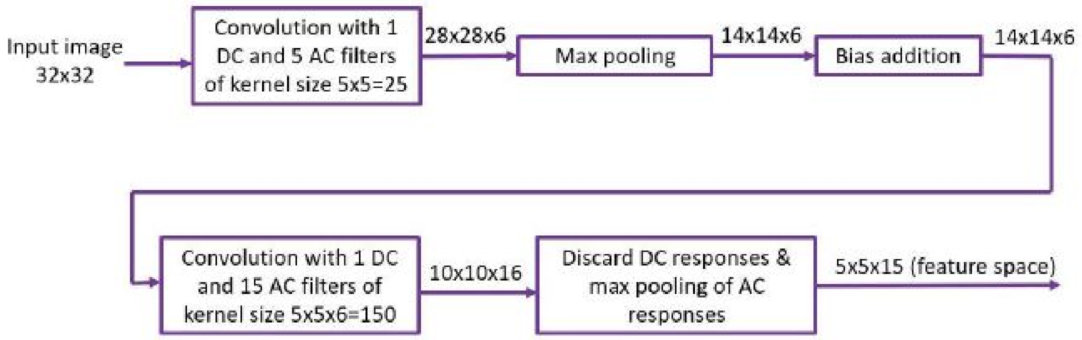

Input image 32x32
Convolution with 1 DC and 5 AC filters of kernel size 5x5=25
28x28x6
Max pooling
14x14x6
Bias addition
14x14x6
Convolution with 1 DC and 15 AC filters of kernel size 5x5x6=150
10x10x16
Discard DC responses & max pooling of AC responses
5x5x15 (feature space)

Figure 3: FeedForward design of Convolutional Layers

Fully Connected Layers in Feed Forward design: In the feedforward design we consider each fully connected layer as a **Least Squared Regressor.** We set up a linear system of equations to find the transformation matrix from high dimensional input space to the output space. Since, there won't be any labels associated with the input features. Inorder to tackle this we use K-means clustering to cluster the classes in Q clusters. The linear system of equations is set up by the following equation.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & w_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & w_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{c1} & a_{c2} & \cdots & a_{cn} & w_c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_c \end{bmatrix}$$

We may say that we can only use one layer between the input features and output labels. But the performance deteriorates directly. So, we generate **pseudo labels** for each hidden layer. Generating the pseudo labels helps us to capture **intra class diversity.** Figure 4 shows the mapping of input to one hot output vector in a new output dimension.

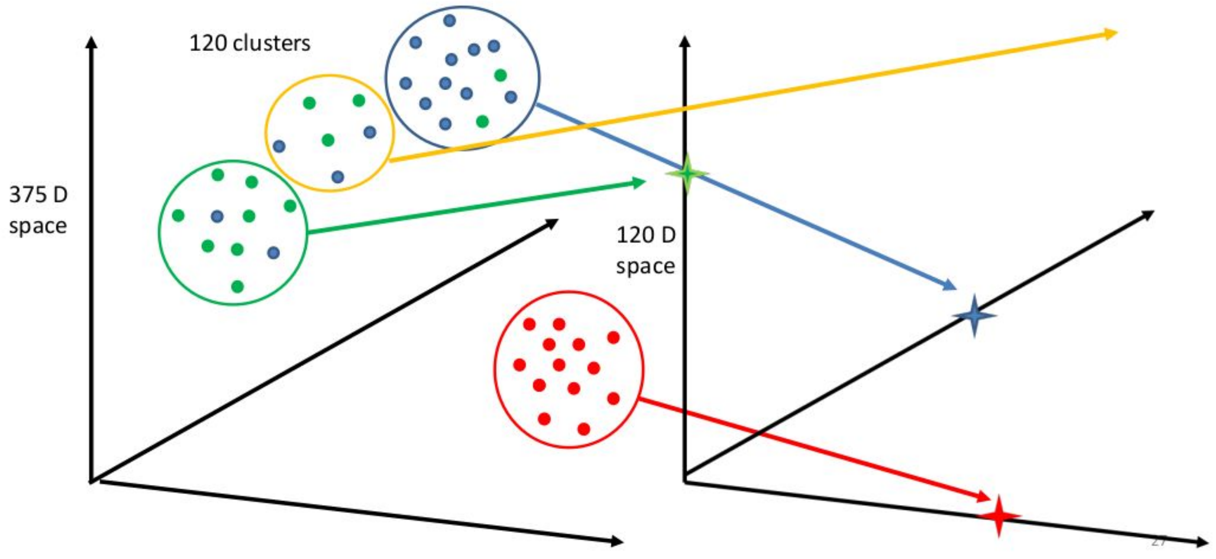


Figure 4: LSR setup and Hard Pseudo Labels

Flow chart of Saab Transform is shown in Figure 5.

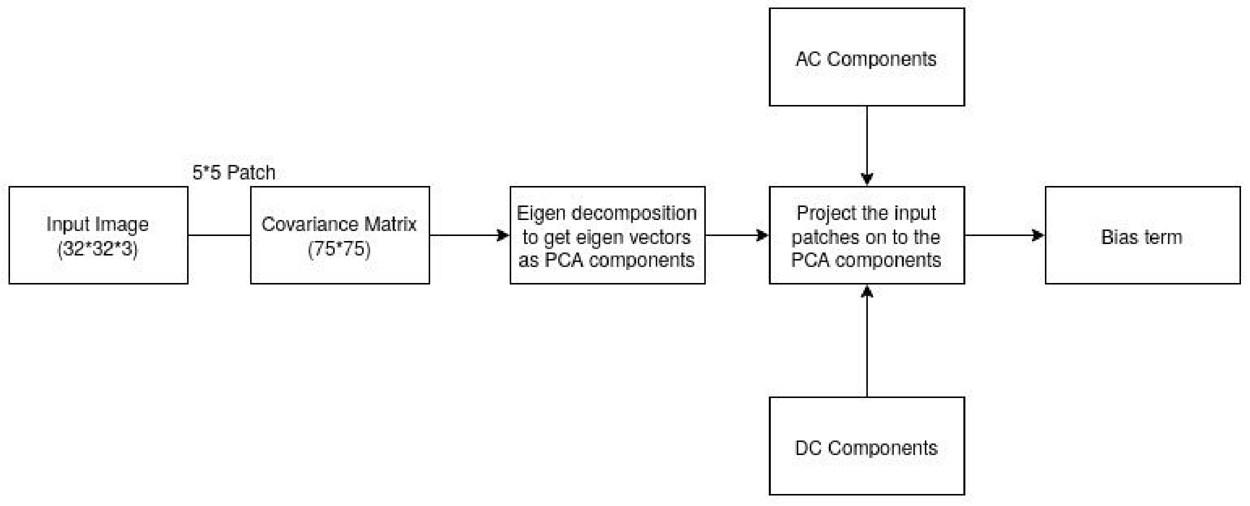

Input Image (32*32*3)
5*5 Patch
Covariance Matrix (75*75)
Eigen decomposition to get eigen vectors as PCA components
AC Components
Project the input patches on to the PCA components
DC Components
Bias term

Figure 5: Flowchart of Saab transform.

Similarities between FF-CNN and BP-CNN (Feedforward vs Backpropagation CNN):

1. The dimension of input space decreases gradually after each layer.
2. Pooling reduces the spatial dimension in both the approaches.
3. Both the approaches are vulnerable to adversarial attacks.
4. Near-to-far neighborhood construction is done in both BP-CNN and FF-CNN to extract the features from local regions to global regions.

Differences between FF-CNN and BP-CNN (Feedforward vs Backpropagation CNN):

FeedForward CNN	Back Propagation CNN
The main idea behind FF-CNN is to learn the filter parameters through local statistics of the image using PCA in a feedforward way	The main idea behind Back Propagation CNN is to optimize the filter parameters using a cost function through back propagation
The interpretability is easy because there is transparency in the process	Interpretability is difficult because the system is considered to be a black box
It's based on KLT - Transform, Linear Algebra and Statistics	It's based on non-convex optimization
The training complexity is low	Training complexity is very high
The architecture is not end-to-end	End-to-end architecture
The Feature extraction layer (Convolution layer) and Fully connected layers are not tightly coupled	All the modules in the network are tightly coupled
This is more like representing an image in a transformed space.	This is basically finding match filters to extract features from the image
Can use the same architecture for different tasks.	Different tasks require different architecture and hyper-parameter selection.

Performance is slightly low compared to BP-CNN	Performance is better compared to FF-CNN

2) Successive Subspace Learning: Based on the FeedForward Convolutional Neural Network approach, Kuo et.al proposed a Machine Learning methodology for object recognition and classification. The concept is based on three key ideas,

a) Successive expansion of the neighborhood (near-far).
b) The feature extraction stage is unsupervised where the dimension of the image is reduced.
c) Supervised classification of extracted features using Least Squares Regression.

Successive Subspace Learning (SSL) is developed by keeping the downsides of CNN in mind. So, SSL is obviously mathematically more profound and interpretable than CNN. Also SSL prevails in terms of scalability and robustness.

Subspace learning is based on representing the data concisely to get accurate decisions on the training data. So, SSL uses a subspace to represent the feature space of a certain class of object. SSL is a continued venture of Saab transform explained in the previous section, where the parameters of convolutional layers are obtained in a feedforward way using multi-stage saab transform and parameters of Fully Connected layers are determined by Least Square Regression (LLSR). The new design based on a feedforward way of finding the parameters is named **PixelHop.** PixelHop method contains a set of independent modules. So, unlike CNN it is **“not a network any longer”.** Figure 5 shows the block diagram of PixelHop method.

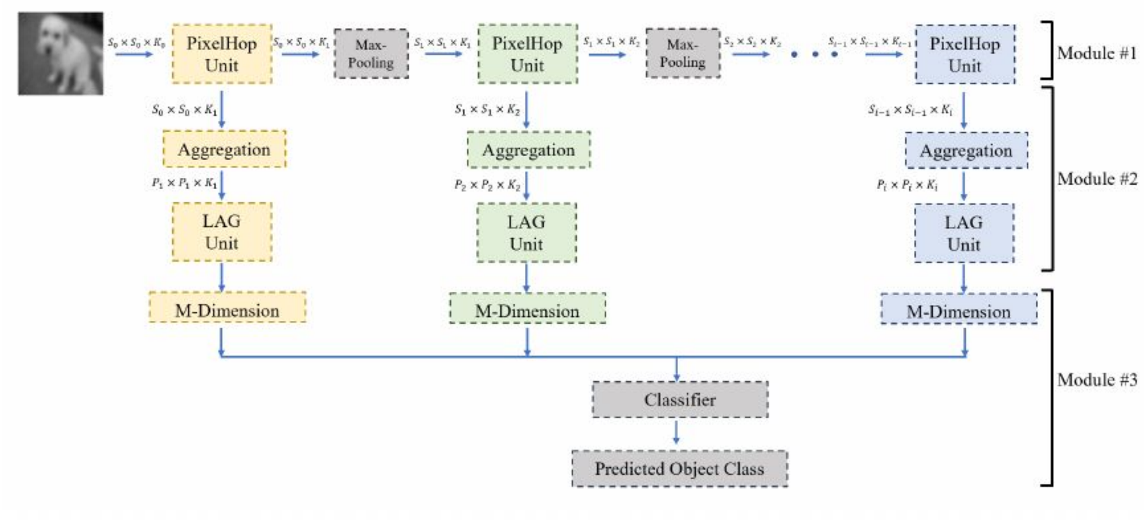


Figure 5: PixelHop Method

Before diving into the details of each module, let us understand the key differences between Deep Learning approach and SSL approach.

Comparison between SSL and DL: While the high level overview of both the methods are the same, SSL design is fundamentally different from DL. Some of the similarities between SSL and DL are listed below,

a) Both DL and SSL collect the attributes in a growing neighborhood fashion.
b) In both the methods, spatial dimension will diminish while the spectral dimension will increase.
c) Both the methods use pooling to reduce the spatial dimension.

Having discussed the high level similarities between SSL and DL, let us highlight some key differences between these two methodologies.

a) **Deep Learning is a parametric model,** in which the network architecture is fixed and usually colossal. This leads to increased model parameters which are typically more than the number of training samples. While this gives DL an edge over the accuracy, it takes huge hardware resources.

   On the other hand **SSL is a non-parametric model,** where the size of the model can be flexibly varied based on the task, number of training sets. So, **model expandability** is the biggest advantage of SSL over DL.

b) A DL model which performs well on a particular data set with a fixed number of classes, will fail to give the same performance on other datasets. Even if an additional class of images is added to the same training set, the model needs a revision. But since SSL is a non-parametric model, we can tune the number of saab filters based on the requirement and also we can tune the LLSR parameters if the additional classes are added. So, **incremental learning** is possible in SSL but not in DL.

c) DL is an **end-to-end tightly coupled** model where each block is influenced by other blocks. So, it's an easy way to design adversarial attacks on DL. But SSL doesn't employ network architecture which makes it more flexible and robust than DL models.

d) Many researchers attribute Deep Learning models as a **black box** because of non-convex optimisation. The reason is, Deep Learning models are **mathematically intractable**. While on the other hand, the mathematics behind SSL is **transparent** and easily interpreted.

e) Deep learning is an **iterative method** which employs back-propogation to optimise a cost function. But SSL is a **one-pass approach** where parameters are determined in a feed forward way.

f) Because of back propagation, the **complexity of training** is more in Deep Learning. But the SSL training process is simple as it employs a feed forward methodology.

g) The convolutional layers used in both DL and SSL look the same in a high level overview. But, fundamentally they are different. Convolution layers in CNN are used to find the match filters, while Convolutional layers in SSL are just a projection of input patches onto the subspace (Linear Transformation of Input Space to Low dimensional subspace).

h) Multi tasking is a bit of a problem in Deep Learning. A single architecture often fails to generalize on different tasks. Usually, joint cost functions are defined to achieve multitasking, but it's not promising. But in SSL, because of the unsupervised feature extraction stage, multitasking is very easy.

i) **Need of huge number of data** is considered to be a conundrum in many of the Deep Learning based approaches. In medical image processing, the data is often less compared to other vision datasets. Since, the model size of DL models are huge, it requires more training images. But the luxury of dataset is not always true in many fields. So **data augmentation** needs to be done. But SSL can perform equally well on small amounts of training data.

j) DL is v**ulnerable to adversarial attacks** because the layers are tightly coupled and people can fool deep networks by adding minute perturbations to the training data. SSL is **not tightly coupled** and there exists a clear firewall between classification module and feature extraction module which makes it difficult for the attackers.

Functions of different modules in SSL framework:

Module 1 - A set of PixelHop units in cascade: The PixelHop unit does neighborhood construction in near-to-far fashion. It consists of 2 modules, one is for neighborhood construction and another one is for subspace approximation (saab transform). We choose 8 neighborhoods of a pixel ($N_i = 8$) but including the particular pixel makes it $N_i + 1$. So we generate a $9 * K_i$ spectral components in each PixelHop unit, which grows rapidly in cascade effect, so we use saab transform to reduce the spectral dimension growing. Figure 6 shows the block diagram of neighborhood construction and saab transform.

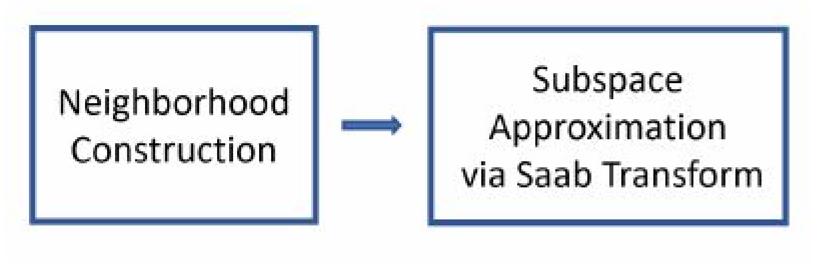

Neighborhood Construction
Subspace Approximation via Saab Transform

Figure 6: Neighborhood construction and Saab transform

Adding a subspace approximation unit reduces the spectral dimension, but the spatial dimension keeps on increasing after each cascading layer. So, $2*2$ max pooling layer is added to reduce the spatial dimension.

Module 2 - Aggregation and dimension reduction using supervised learning techniques (Label Assisted Regression - LAG):

The cascaded PixelHop units are connected to each other i.e, the output of i^{th} pixel hop unit is connected to the input of $(i+1)^{th}$ PixelHop unit. But the spatial dimension will keep growing, so we add a max pooling layer to downsample the spatial dimension. To capture the diverse set of features, we take different response values like mean, minimum and maximum values in a non overlapping patch of the image. The output of this is denoted by P_i. This will be combined to form a 1D feature for each image and the output is given to the LAG unit for classification tasks. Till, module 2 everything will be unsupervised. After LAG unit, we use labels to further proceed with the classification.

Figure 7 shows the block diagram of Module 2.

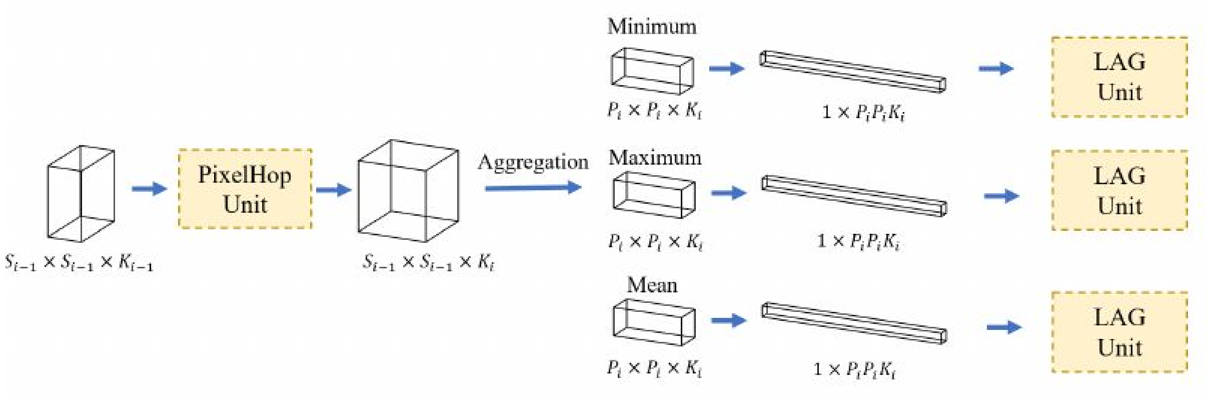

$S_{i-1} \times S_{i-1} \times K_{i-1}$
PixelHop Unit
$S_{i-1} \times S_{i-1} \times K_i$
Aggregation
Minimum
$P_i \times P_i \times K_i$
$1 \times P_i P_i K_i$
LAG Unit
Maximum
$P_i \times P_i \times K_i$
$1 \times P_i P_i K_i$
LAG Unit
Mean
$P_i \times P_i \times K_i$
$1 \times P_i P_i K_i$
LAG Unit

Figure 8: Block Diagram of Module 2

Module 3 - Feature Concatenation and Classification: We concatenate all the features from all pixel hop units to create a feature matrix of size $M*I$. From here onwards, it's treated as a **classical pattern classification problem**. We can use any classification method like **SVM, Random Forest, Naive Bayes etc**. The role of the LAG unit is explained in detail in the below section. So module 3 is analogous to the fully connected layer.

Neighborhood construction in PixelHop method: As explained in the previous section, the neighborhood construction in a PixelHop unit. We elaborate and discuss the improvements in PixelHop++ method. As we know the pixel hop unit first implements neighborhood construction and then uses Saab transform to reduce the growing spectral dimension. So, the i^{th} PixelHop unit takes the attributes from the previous layer whose spectral dimension is K_{i-1} to construct a neighborhood pixel N_i which forms neighborhood unions. So if we do this operation across all PixelHop units, we get a feature vector of dimension $K_0 9^i$. Inorder to check the spatial dimension, max pooling is applied separately on each spectral component. Figure 9 shows the block diagram of PixelHop system.

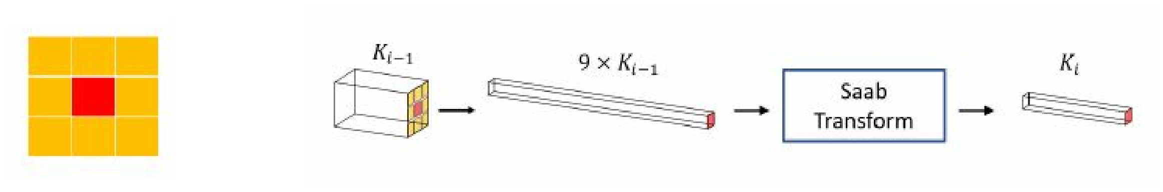


Figure 9: Neighborhood Construction

Comparison between Neighborhood construction between PixelHop and PixelHop++ method:

The neighborhood construction in both PixelHop and PixelHop++ are fundamentally the same except for few changes.

a) In PixelHop++, three PixelHop units are used in cascade.
b) All the PixelHop use channel wise (c/w) saab transforms.
c) A tree based representation of features.

Channel wise (c/w) Saab transform: It is observed that the correlation between the spectral component is less compared to the spatial components. So, a Saab transform is approximated as a tensor product of transforms along spatial domain and spectral domain. By using these two observations, we can reduce the size of the Saab transform kernels. Instead of using 3D tensors as anchor vectors, we can use 2D tensors. Figure 10 shows the difference between c/w Saab transform and Saab transform.

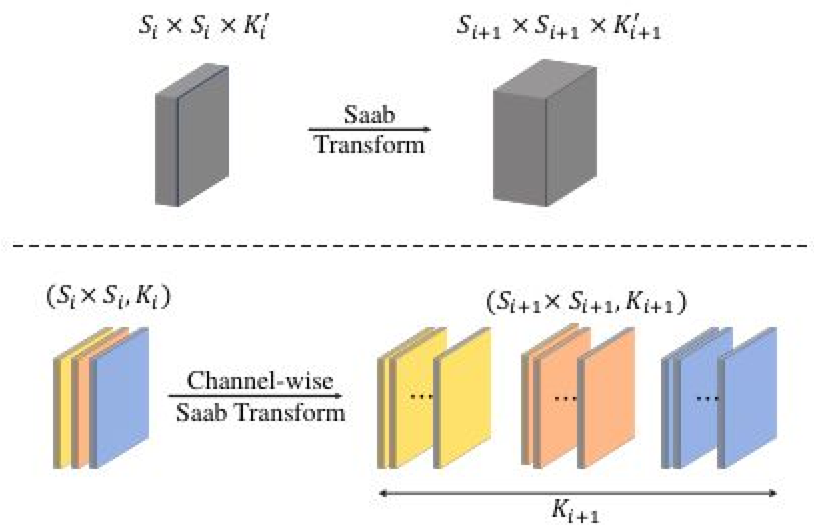

$S_i \times S_i \times K_i'$
$S_{i+1} \times S_{i+1} \times K_{i+1}'$
Saab Transform
$(S_i \times S_i, K_i)$
$(S_{i+1} \times S_{i+1}, K_{i+1})$
Channel-wise Saab Transform
K_{i+1}

Figure 10: Difference between c/w Saab transform and Saab transform

In addition to the c/w Saab transform, a novel approach called **"Tree based representation of features".** In this method, we stop growing the tree i.e., Saab decomposition of high frequency components. Instead we continue the decomposition of low frequency i.e., we reduce the spatial dimension of lower frequency components.

Tree decomposition criteria:

a) Higher energy nodes have lower frequency components and tend to have wider spatial correlation.
b) Lower Energy nodes have higher frequency components and tend to have narrower spatial components.

Figure 11 shows the tree decomposition.

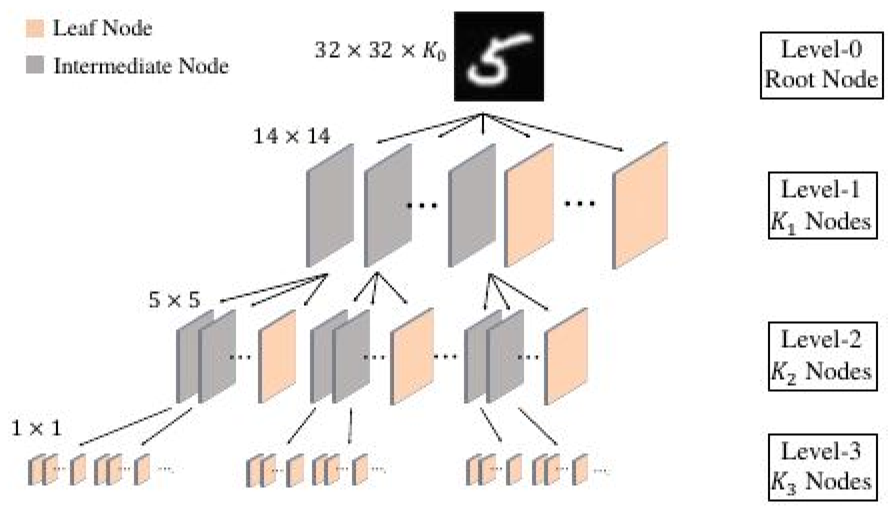


Figure 11: Tree Decomposition

Label Assisted reGression (LAG): The Least Square Regression introduced in Saab transform uses pseudo labels by clustering samples from the same class to account i**ntra class variability**. This method uses **hard labeled outputs** (one-hot vector) to train the regression matrix. But the LAG unit in PixelHop method uses **soft labeled outputs** which gives the probability in which the sample belongs to a particular class. The training of a regression matrix is the same in both the cases but defining the probability vector is a new thing to focus on. Figure 9 shows the example of intra class variability which needs to be considered.

In the Figure, all the images belong to a single class but due to the orientation of the camera while taking pictures, we might get different feature vectors far apart. So, this will be taken care in LAG unit

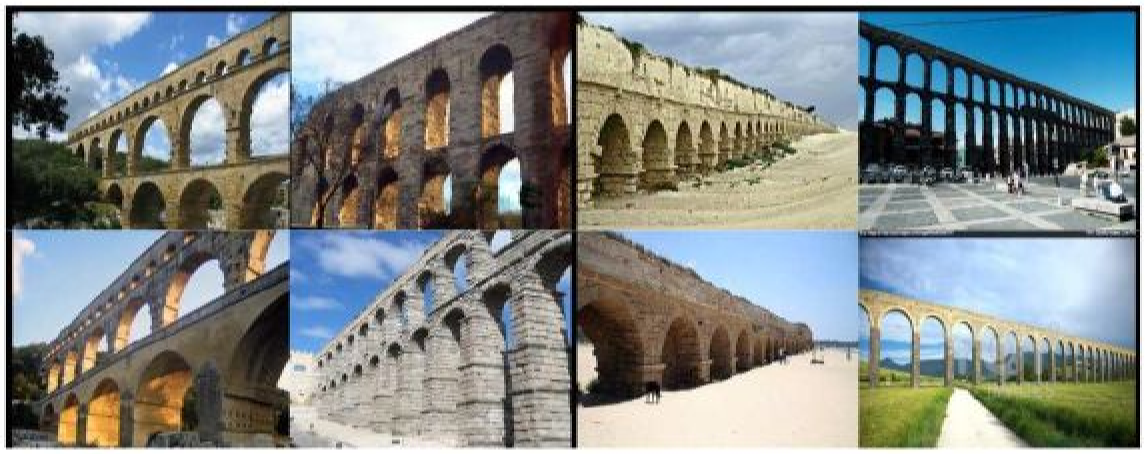


Figure 9: Intra class variability

We have explained the neighborhood construction in the previous section. In this section, we discuss **"Cross Entropy guided feature selection"** and LAG unit. LAG unit is common in both PixelHop and PixelHop++ methods, but cross entropy based feature selection is a novel idea of PixelHop++ method.

The output of module 1 (PixelHop) unit have features vectors of dimension $1 * (P_i * P_i * K_i)$ after aggregation from the PixelHop unit i for a single training sample. We will have n such samples making the dimension $n * (P_i * P_i * K_i)$ After this, we calculate the cross entropy of each feature using the original labels using the equation given below,

$$L = \sum_{j=1}^{J} L_j , \quad L_j = -\sum_{c=1}^{M} y_{j,c} log(P_{j,c})$$

where, $y_{j,c}$ is the original label of sample j and it is set to one if it belongs to class c, $P_{j,c}$ is the probability which defines sample j belonging to class c. So, we pick top 1000 or top 50% features based on cross-entropy values. Lesser the cross entropy value, higher the discriminant power of the feature. By adding the cross entropy based feature selection we can reduce the dimension of new feature space by a significant amount which inturn reduces the model size.

After selecting the features with higher discriminant power we have to pass these features to the LAG unit to train the regression matrix. We can describe the operations of LAG unit in 3 steps,

a) Clustering the samples which belong to the same class to create a subspace for the particular object class and find the centroid of each subclass. This step is to account intra-class variability.
b) Calculating a probability vector based on the euclidean distance between centroids and the samples.
c) Setting up a linear Least Square Regression matrix by solving the linear system of equations.

For the first step, we use K-means clustering and apply it to the samples from the same class. We cluster the input samples from every class into Q clusters and if we have C classes, then we will have $Q * C$ clusters on a whole. So, we define the probability of an input sample of class c belonging to one of the q clusters is given by,

$$P(x_c, C_{c',l}) = 0 \; if \; c \, != \, c'$$

$$P(x_c, C_{c',l}) = \frac{exp(-\alpha d(x_c, C_{c',l}))}{\sum_{l=1}^{L} exp(-\alpha d(x_{c,} C_{c',l}))}$$

Here d is the euclidean distance and α is the hyper-parameter to be chosen. Figure 10 shows the clustering of samples from each class.

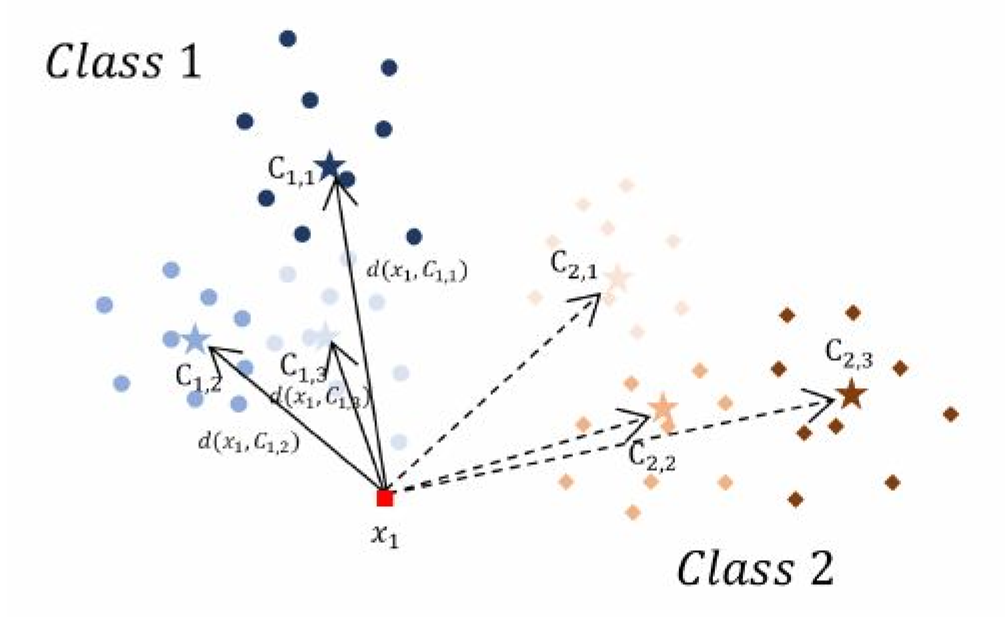


Figure 10: Clustering of samples from each class.

After clustering, we set up linear LSR equations to map the input samples with corresponding probabilities. The equation is shown below,

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & w_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & w_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{Mn} & w_M \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1(\mathbf{x}) \\ \vdots \\ \mathbf{p}_j(\mathbf{x}) \\ \vdots \\ \mathbf{p}_J(\mathbf{x}) \end{bmatrix},$$

Advantages of using LAG unit: Conceptually LAG unit is nothing but solving a linear system of equations to determine the coefficients of a regression matrix. This on the higher level replaces the fully connected layers of CNN. Let us discuss the advantages of the LAG unit over fully connected layers.

a) LAG unit connects the features extracted from PixelHop units in an unsupervised way to the specific task.
b) More specifically, LAG units bridge the gap between general features to the features corresponding to specific tasks.
c) Unlike back propagation, instead of optimizing a cost function, LAG units learn the parameters in one pass. This reduces time significantly.
d) LAG accounts for intra class variability. It helps to handle such cases more efficiently in an easier way than back propagation.

2) CIFAR-10 Classification using Successive Subspace Learning (SSL):

1) Train accuracy, Training time and Model Size:

The PixelHop++ model is trained on CIFAR-10 dataset dataset using **GPU** provided by **Google Collab** with a **RAM size of 25GB.** The training time is calculated based on the log file generated by Google Collab for individual cells. The calculation formula is given below,

$$T_{train} = \sum_{c=1}^{C} t_c$$

where t_c corresponds to the training time associated with each cell corresponding to the training code. Table 1 shows the training time for each module.

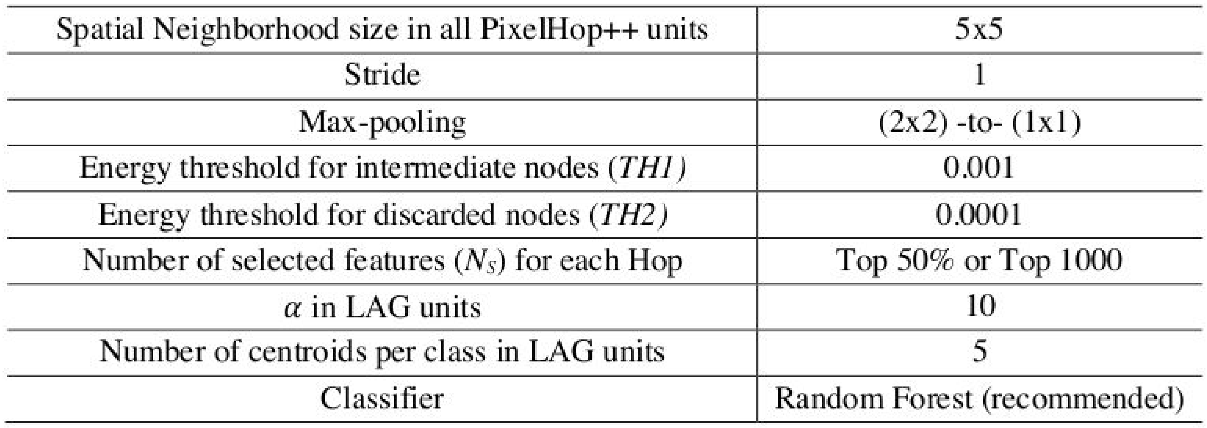
Module	**Layer 1**	**Layer 2**	**Layer 3**	**Total**
PixelHop	647.2s	499s	161.8s	1308s
Cross Entropy	3384.04s	1228s	513.3s	5125.34s
LAG	1678s	1020s	187s	2885s
RF-Training	NA	NA	NA	131.086s

$$T_{train} = 1308 + 5125.34 + 2885 + 131.086 = 9449.426s$$

which is approximately equal to **157.49 minutes.**

Training Accuracy: After concatenating the feature vectors of each image from the LAG unit, we create a feature matrix of dimension $(50000,\ 150)$ and train this using a Random Forest Classifier.

The hyper-parameters of the Random Forest classifier are tuned by trial and error method. Although, performing cross validation is a better option to get highest accuracy, but due to time constraint could't implement it.
Figure 11 shows the hyperparameter selection for the training model.



Spatial Neighborhood size in all PixelHop++ units	5x5
Stride	1
Max-pooling	(2x2) -to- (1x1)
Energy threshold for intermediate nodes (*TH1*)	0.001
Energy threshold for discarded nodes (*TH2*)	0.0001
Number of selected features (N_S) for each Hop	Top 50% or Top 1000
α in LAG units	10
Number of centroids per class in LAG units	5
Classifier	Random Forest (recommended)

Figure 11: Hyperparameter Selection

Chosen Hyperparameters for RF classifier:

n_estimators = 200, max_depth = 8, random_state = 0.

where, n_estimator corresponds to the number of trees, max_depth corresponds to the depth of leaf nodes to stop growing.

For the above parameter setting, the train accuracy of **93.7%** is noted. Although this is not the highest accuracy but for the tuned parameters, we got the highest train and test accuracy.

So, **Train accuracy = 93.7%**

Model Size: The model size is calculated based on the block diagram shown in Figure 12.

Note: In cross entropy based feature selection, top 50% features with lowest entropy values are selected.

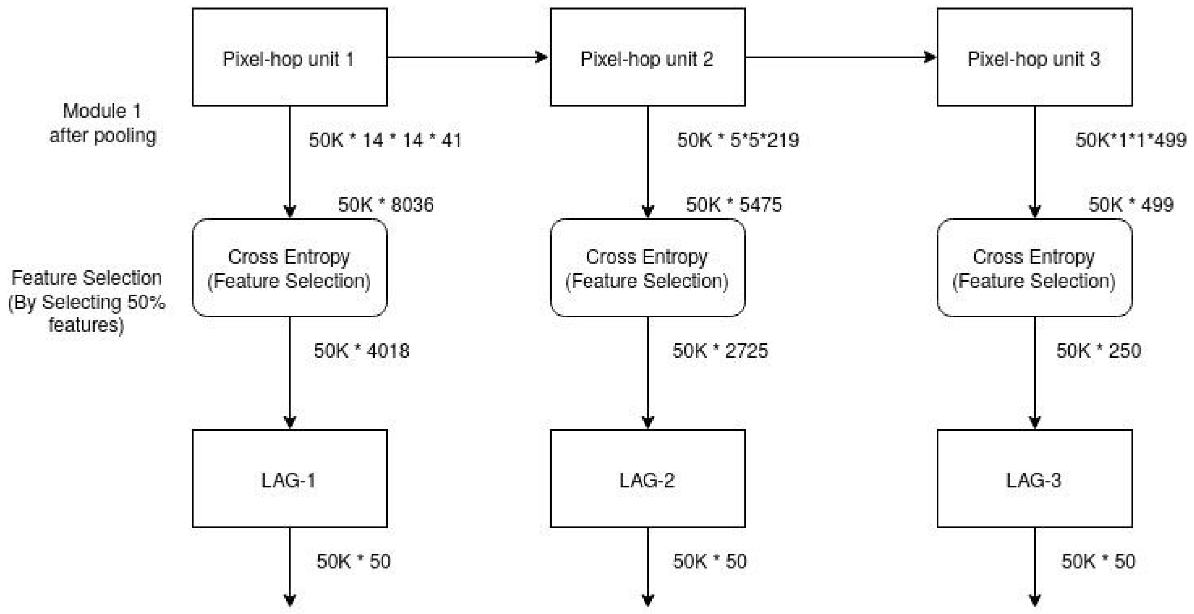


Figure 12: Block diagram based on results.

As we can see from the above figure, we can calculate the model size based on the result obtained from each module.

Note: 5*5 corresponds to 5*5 window sizes we use while neighborhood construction.

Module	Layer 1	Layer 2	Layer 3	Total
PixelHop unit	5*5*3*41 = 3075	5*5*219 = 5475	5*5*499 = 12475	21025
LAG	50*(4018+1) = 200950	50*(2725+1) =136300	50*(250+1) =12550	349800
RF classifier	NA	NA	NA	NA

Total Number of Parameters = 370825

2) Test Accuracy: After training the model parameters using 50000 training images, we test the model using 10000 test images. The resulting accuracy is **64.708%** for the chosen hyperparameters of the random forest classifier.

So, **Test Accuracy = 64.708%.**

3) Weak Supervision: We train the module 1 with the same number of images, but we keep on reducing the number of training images from module 2. This gives us an idea of how the number of data is related to the test accuracy. Table 2 shows the resulting accuracy and number of images.

Training Images	50000	12500	6250	3125	1563
Test Accuracy	0.6470	0.4438	0.2739	0.1673	0.0841

Figure 13 shows the plot of Test Accuracy vs Number of Training Images.

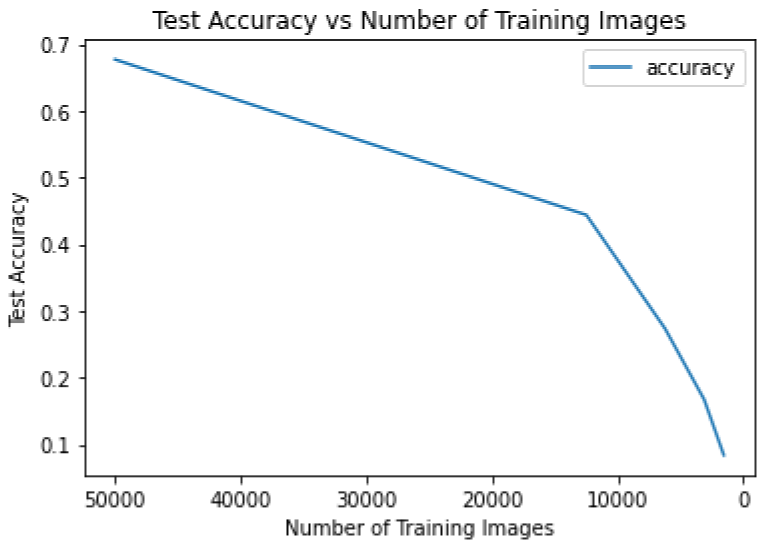
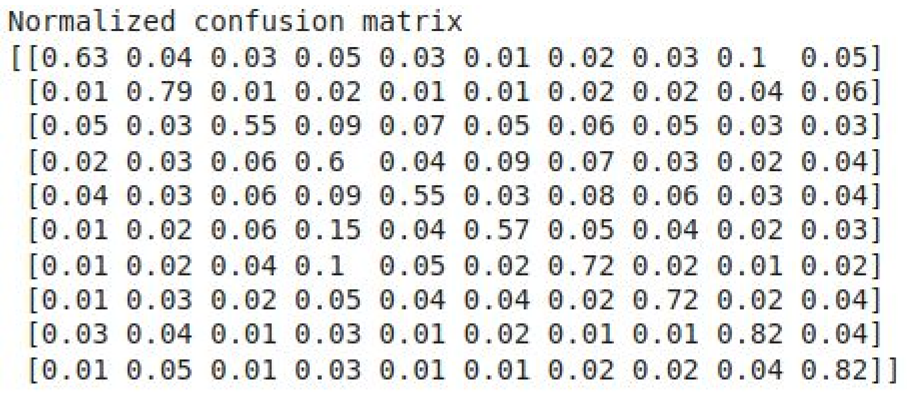


Figure 13: Test Accuracy vs Number of Training Images

Discussion on Results: As we know, CNNs are parametric models and SSL is a non-parametric model. So, the model parameters will be constant regardless of the number of training images in CNNs. But SSL being a non-parametric model, the model parameters decrease significantly.

Also, the accuracy fall is nominal after reducing the training data by ¼, ⅛. This is because of the **fast convergence of Saab filters.**

2) Error Analysis: Error analysis is performed to analyse the performance of the mode and to improve the accuracy. This can be done by calculating the confusion matrix. Figure 14 shows the confusion matrix.



```
Normalized confusion matrix
[[0.63 0.04 0.03 0.05 0.03 0.01 0.02 0.03 0.1  0.05]
 [0.01 0.79 0.01 0.02 0.01 0.01 0.02 0.02 0.04 0.06]
 [0.05 0.03 0.55 0.09 0.07 0.05 0.06 0.05 0.03 0.03]
 [0.02 0.03 0.06 0.6  0.04 0.09 0.07 0.03 0.02 0.04]
 [0.04 0.03 0.06 0.09 0.55 0.03 0.08 0.06 0.03 0.04]
 [0.01 0.02 0.06 0.15 0.04 0.57 0.05 0.04 0.02 0.03]
 [0.01 0.02 0.04 0.1  0.05 0.02 0.72 0.02 0.01 0.02]
 [0.01 0.03 0.02 0.05 0.04 0.04 0.02 0.72 0.02 0.04]
 [0.03 0.04 0.01 0.03 0.01 0.02 0.01 0.01 0.82 0.04]
 [0.01 0.05 0.01 0.03 0.01 0.01 0.02 0.02 0.04 0.82]]
```

Figure 14: Normalized Confusion Matrix

Figure 15 shows the heat map.

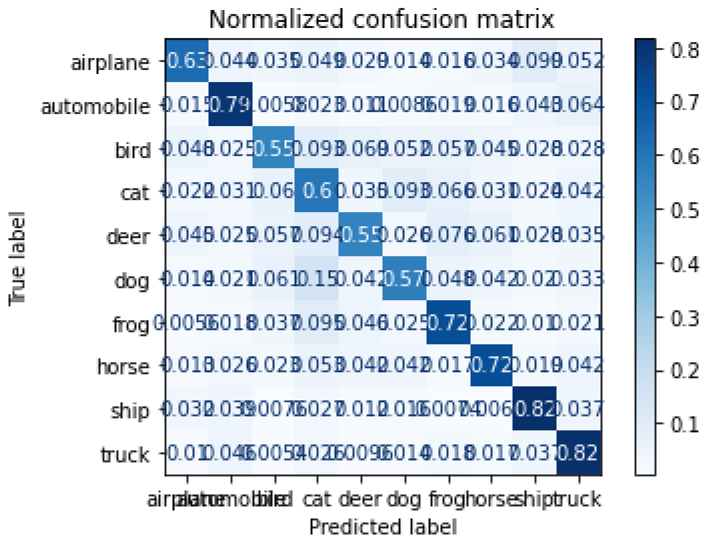


Figure 15: Heat Map

Note: There is some issue with Google collab. While viewing the in window, the confusion matrix is clear but when I download the image, it's getting blurred. Please refer to the raw confusion matrix above.

Discussion: As we can see from the confusion matrix, Class **Ship** and **Truck** have highest accuracy whereas for classes **Deer** and **Bird** the accuracy is low.

Also, if we can observe carefully, both **deer and dog images are being predicted as Cat.** Almose 10% of the data of dog and cat each has been predicted wrongly among cats in these classes. Let us examine the reason with some images. Figure 16 shows the images with labels and it helps us to analyze the results further.

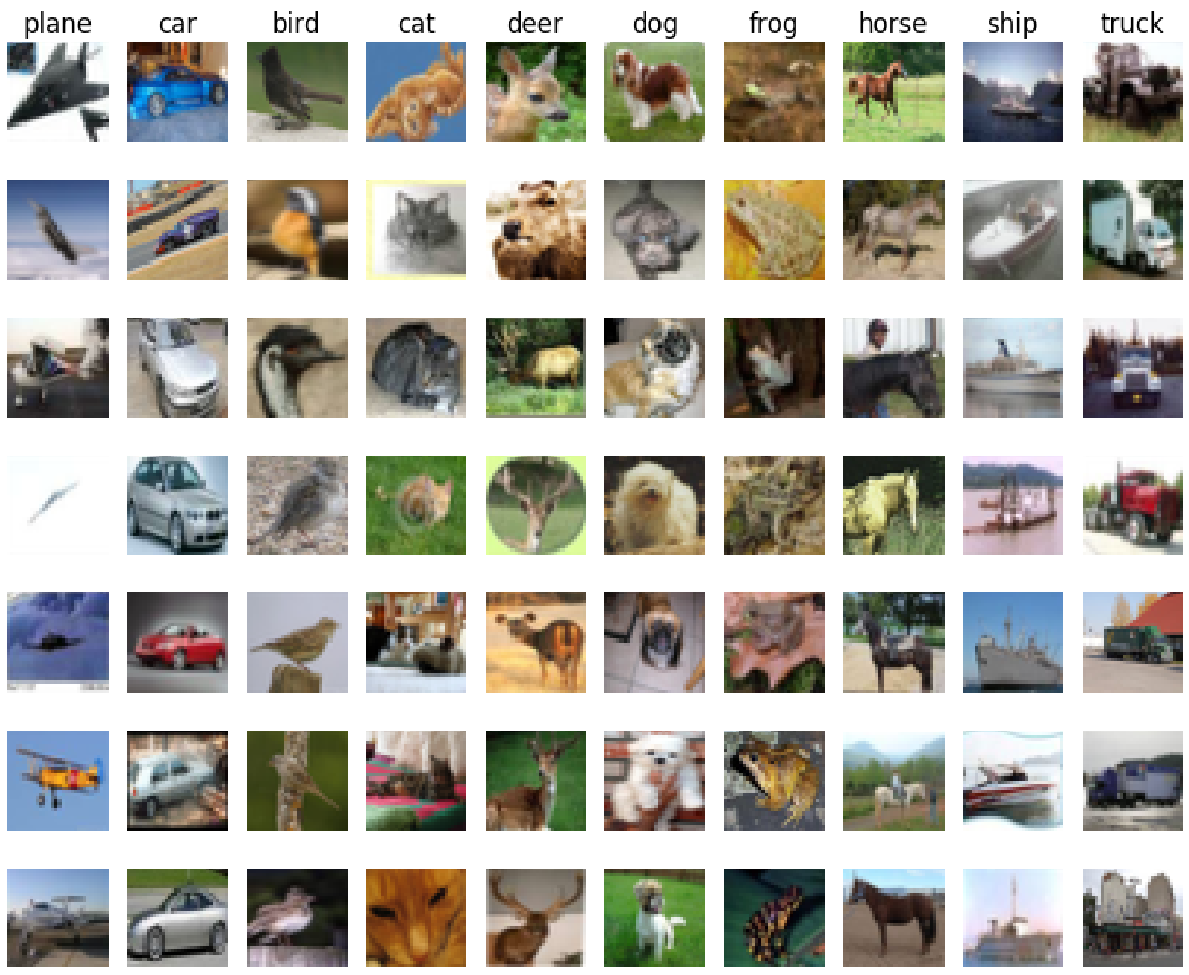


Figure 16: CIFAR-10 Images with labels

As we can observe from the image, the resolution and also the spatial components of both the classes like ears, eyes are somewhat misleading. We can consider them as close classes. In some cases, even birds are being misclassified as cats. Around 5% of horse images are being misclassified as dogs. From this we can infer that the animal classes are often misclassified among themselves rather than material classes (like ship, aeroplane etc) because of similar spatial features i.e., the feature vectors will be very close and make it harder for the classification.

Ideas to Improve the accuracy: We can improve the accuracy by changing functionalities of some of the modules.

a) **Instead of cross entropy based feature selection we can use dimensionality reduction techniques like PCA**: In the module 2 we use cross entropy based feature selection to reduce the number of features based on the cross entropy values. But instead of removing the features, we can use **PCA or Fisher's Discriminant Analysis** where the new feature dimension will be the linear combination of the original features. Unlike the feature selection method PCA projects the data in high dimension to a lower dimension subspace. This can avoid features being discarded and may improve the accuracy.

b) **Using K-Fold Cross Validation for picking hyperparameters of the classifier:** Instead of guessing the model parameter we can use K-Fold cross validation to get the hyperparameters which can get highest accuracy.

c) **Using Statistical Classification Methods:** We can try implementing Bayesian classification methods instead of SVM/Random Forest. Since the input image classes form a certain distribution, we can predict based on the distribution parameters.

d) **Introducing Aggregation Layer:** Using aggregation techniques as introduced in the class have potential to improve the accuracy.

e) **Data Augmentation:** Augmenting the data will give access to a wide variety of data with increased training images. So, augmentation can also boost accuracy.

References:

1. C.-C. Jay Kuo, “Understanding convolutional neural networks with a mathematical model,” the Journal of Visual Communications and Image Representation, Vol. 41, pp. 406-413, November 2016.
2. C.-C. Jay Kuo and Yueru Chen, “On data-driven Saak transform,” the Journal of Visual Communications and Image Representation, Vol. 50, pp. 237-246, January 2018.
3. C.-C. Jay Kuo, Min Zhang, Siyang Li, Jiali Duan and Yueru Chen, “Interpretable Convolutional Neural Networks via Feedforward Design,” the Journal of Visual Communications and Image Representation, Vol. 60, pp. 346-359, 2019 (or arXiv 1810.02786).
4. M. Zhang, H. You, P. Kadam, S. Liu, C.-C. J. Kuo, “PointHop: an explainable machine learning method for point cloud classification,” arXiv preprint arXiv:1907.12766 (2019), to appear in IEEE Trans. on Multimedia.
5. Yueru Chen and C.-C. Jay Kuo, “PixelHop: a successive subspace learning (SSL) method for object classification,” arXiv preprint arXiv:1909.08190 (2019), to appear in the Journal of Visual Communications and Image Representation.