Vegetation Classification Using Hyperspectral Images

Fengjunyan Li, Kazim Ergun, Osman Cihan Kilinc, Yuming Qiao

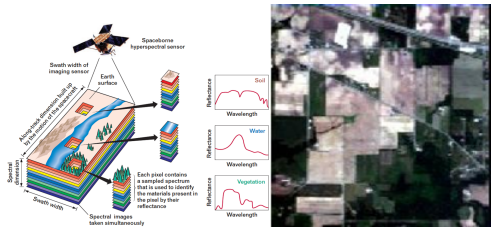
***Abstract*— Every object has its unique spectral 'signature' over the electromagnetic spectrum, and this can be captured using hyperspectral sensors. In this project, we use these signatures for discriminating different types of plants. It has been shown that machine learning algorithms are very powerful at learning patterns. They have already been utilized for classification tasks in various fields. Indian pines dataset[1] is used for testing and other measurements. Treating the hyperspectral bands in the images as features and each pixel as a sample, we classify plants with convolutional neural networks(CNN) and support vector machines(SVM). Furthermore, we optimize CNN to prevent overfitting, accelerate inference, and reduce the resources it uses with respect to memory, battery and computational power. The results demonstrate that CNN is very successful in hyperspectral image classification tasks and optimizations further increase its accuracy. We achieved 83.9% test accuracy using SVM with Polynomial kernel, and successfully achieved 99.2% with CNN.**

I. INTRODUCTION

Hyperspectral imaging, like other spectral imaging, collects and processes information across the electromagnetic spectrum, usually in visible, near infrared, and short-wave infrared wavelengths. Recently, with the development of hyperspectral sensors, it has become possible to go beyond traditional RGB images and capture hundreds of spectral bands sampled with narrow wavelength intervals. Therefore, taking advantage of contiguous narrow bands, these hyperspectral sensors enabled the study of the chemical properties of scene materials remotely for the purpose of identification, detection, and chemical composition analysis of objects in the environment. Hence, hyperspectral images captured from earth observing satellites and aircraft have been increasingly important in agriculture, environmental monitoring, and urban planning.

The fundamentals of hyperspectral imaging are based primarily on the interaction of light with matter. When a photon is incident on a surface of a medium, energy can be absorbed, transmitted and/or reflected by that surface, with a wavelength dependency determined by the material properties. The ratio of the energy reflected or scattered by the surface to the incident energy is termed as the reflectance, and measured by the hyperspectral sensors. For a particular object, the reflectance spectrum shows characteristic bands induced by the constituent materials. Therefore, spectral reflectance provides substantial information about the material properties.

All authors are graduate students in University of California San Diego, Electrical and Computer Engineering



(a) Hyperspectral Images (b) Satellite Image of Indian Pines

Fig. 1: Hyperspectral Images and Indian Pines Dataset

In this project, we utilize the spectral reflectance information to classify plant types as an application of precision agriculture. Specifically, we use hyperspectral images captured from satellites and use machine learning techniques to discriminate crops in a plantation area. We use Indian Pines Hyperspectral Dataset[1] gathered by AVIRIS Sensor. For classification, we have used Convolutional Neural Networks(CNN)[2] and Support Vector Machines(SVM) with dimensonality reduction methods Principle Component Analysis(PCA) and Nonnegative Matrix Factorization(NMF). In short, the input to our algorithm is an a hyperspectral image with size $145 \times 145 \times 224$ seen in figure 1 (b). We then use a SVM and CNN model to output a predicted vegetation class for each pixel.

II. RELATED WORK

In their work Delalieux et al.[3] used hyperspectral data to detect apple scab caused by disease. The study involved the identification of infected trees and selection of wavelengths best suited for classifying the infected leaves from those of the healthy leaves. The spectral data were analyzed using methods as LDA, logistic regression analysis (for each wavelength), partial least squares logistic discriminant analysis, and tree-based modeling for classifying the infected leaves classification.

Lee et al.[4] worked on the detection of greening in citrus plantation using hyperspectral images. He used the spectral angle mapping and spectral feature fitting classification techniques. However, the author was not able to achieve high classification accuracies due to a large variability within the data.

Begum Demir and Sarp Erturk[5] used Relevance Vector Machine. They propose that RVM has similar strategy with SVM but require fewer relevance vectors. With lower complexity, RVM can achieve similar classification as SVM. In the paper of G. Mercier and M. Lennon,[6] they researched

SVM with spectral-based kernel to classify satellite hyperspectral images. Their method is able to reduce the false alarm by traditional kernel. We will try to use traditional kernels in the SVM of our project to verify the result.

In their work Makantasis et al.[2] show that using CNN, hyperspectral images can be successfully classified. CNN can encode the spectral and spatial features of pixel. The low-to-high hierarchy of features improve the performance of classification greatly. In our CNN implementation we extend and optimize their method with layer pruning and layer compression methods. Although these methods are widely adopted in the industry and research, to our knowledge no other work has been published that applies such optimizations on top of the model Makantasis et al. provided in their work.

III. Dataset Features and Preprocessing

The dataset Indian Pines[1] consists of 145×145 pixels and 224 spectral reflectance bands in the wavelength range $0.4 - 2.5\,\mu m$. There are 16 classes, some of them being from the same crop type but at different stages of growth. In terms of machine learning, each pixel in the images are labeled according to their class and contain 224 features. Plainly speaking, each pixel represents a location, and that location contains 224 spectral bands. For example, the first five spectral bands of a pixel/location can have values such as : 3172, 4142, 4506, 4279, 4782. Data is preprocessed for four reasons: (i) to introduce variability into data, (ii) to strengthen the weak classes, (iii) to reduce the dimensionality of the feature set and (iv) to prepare data for processing.

A. Oversampling

In Indian Pines dataset, there is a huge imbalance between the number of samples of different crop types. For example, as shown in Table I. There are 20 Oats samples whereas the number of Soybean-mintill samples are 2455. To fix this imbalance issue, we oversampled the classes which has substantially less number of samples compared to the others. Without oversampling, the classifier would give more weight towards the classes with high number of samples, which would lead to misclassification in minority classes. In our implementation, we check the number of samples(pixels) per each class and for each class we concatenate samples from that class to data matrix to make up for the difference between classes. Later, we randomly redistribute samples in the dataset.

B. Data Augmentation

To prevent overfitting and train our model for the real world, where images can come with different rotations and different angles, we introduce new data by augmenting the existing data. This increases the variability of data and prevents overfitting. Hence, it increases the accuracy. This achieved by flipping and rotating the images. In our implementation we randomly choose either to flip left-right, up-down or rotate each sample.

TABLE I: 16 Classes of Indian Pines Dataset.

Class	# of Samples
Alfalfa	46
Corn-notill	1428
Corn-mintill	830
Corn	237
Grass-pasture	483
Grass-trees	730
Grass-pasture-mowed	28
Hay-windrowed	478
Oats	20
Soybean-notill	972
Soybean-mintill	2455
Soybean-clean	593
Wheat	205
Woods	1265
Buildings-Grass-Trees-Drives	386
Stone-Steel-Towers	93

C. Dimensionality Reduction

Hyperspectral data is large in size, since information from whole spectrum range is included in a pixel. However, not all the information contained in the data is relevant for the analysis, thus can be discarded without much loss. Moreover, a lot of the bands shows high correlation. Applying dimensionality reduction on our data before classification step helps us reduce the processing time by projecting the data on a smaller sized space without loss of information.

1) Principal Component Analysis: Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors are an uncorrelated orthogonal basis set. More information about PCA can be found in.[7] To show the amount of information kept after dimension reduction, we plotted the variance ratio for different number of principle components in Figure2. We took the 30 top principle components since it is enough to explain the 0.99 of the total variance.

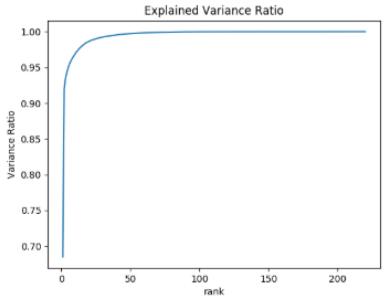


Fig. 2: Variance Explained vs Number of Principle Components

Using the results of the PCA method, we can write $\mathbf{A}\mathbf{x}_i = \lambda_i \mathbf{x}_i$, where x_i are the eigenvectors of $\mathbf{A}$, λ_i are the eigenvalues of $\mathbf{A}$, for any matrix $\mathbf{A}$.

For data given as a set of m vectors $\in \mathbb{R}^n$, $\mathbf{x}_1, ..., \mathbf{x}_m$,PCA method can be formalized as an optimization problem as follows:[8]

$$\begin{aligned} \underset{\mathbf{W}\in\mathbb{R}^{m\times k}}{\text{maximize}} \quad & ||\mathbf{X}\mathbf{W}||_{\mathbf{F}}^2 \\ \text{subject to} \quad & ||\mathbf{W}|| = 1, \end{aligned}$$

where $\mathbf{X}$ denotes the normalized covariance matrix $\mathbf{X} = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^T$ and W is the orthogonal projection matrix.

2) Nonnegative Matrix Factorization: Nonnegative Matrix Factorization (NMF) is a matrix factorization method where a matrix $\mathbf{V}$ is factorized into matrices $\mathbf{W}$ and $\mathbf{H}$, with the property that all three matrices have no negative elements. Similar to PCA, we use NMF as a tool to reduce dimensionality. Difference between PCA and NMF is that while PCA has negative elements, the most rule of NMF is that it does not have any negative elements. Thus, with respect to PCA, NMF has more floating point elements to approximate the matrix as close as possible to the original. A comparison with respect to accuracy between PCA and NMF is provided in the results section. The standard NMF problem can be formulated as follows:

$$\begin{aligned} \underset{\mathbf{W}\in\mathbb{R}^{m\times r},\mathbf{H}\in\mathbb{R}^{r\times n}}{\text{minimize}} \quad & ||\mathbf{V} - \mathbf{W}\mathbf{H}|| \\ \text{subject to} \quad & \mathbf{W}, \mathbf{H} \geq 0, \end{aligned}$$

IV. Classification Methods

A. Support Vector Machines

Support Vector Machine, also known as SVM, is popular for solving problems about classification, detection and regression. In support vector machine, the model constructs a hyperplane or set of hyperplane in higher dimension space. From,[9] we can know the structure of SVM. The hyperplane linear model of SVM can be defined as

$$\mathbf{y} = \mathbf{w}^T \phi(\mathbf{x}) + \mathbf{b}$$

where $\phi(\mathbf{x})$ is transformed feature space. The margin is defined as the smallest distance from the decision hyperplane to the closest point from dataset. In the SVM problem, we are trying to construct the decision boundary hyperplane to maximize the margin with dataset. For each data point, t_i is the target label, where $\mathbf{t}\in\{1,-1\}$. In our case, the problem is not linearly separable, we will use soft-margin SVM and introduce slack variable $\xi_i \geq 0$. ξ allows the misclassification of outline. When $\xi_i > 1$, the data point is misclassified. At the same time, we should have inequality constraint as follows:

$$t_i(w_i^T \phi_i(\mathbf{x}) + b) \geq 1 - \xi_i$$

This is because we need to make $y(x_n) > 0$ for those points that have $t_n = 1$, and make $y(x_n) < 0$ for those points that have $t_n = -1$. The distance from data point x_n to the decision boundary in hard-margin is given by:

$$\frac{t_n y(x_n)}{||\mathbf{w}||} = \frac{t_i(\mathbf{w}^T\phi(\mathbf{x}) + \mathbf{b})}{||\mathbf{w}||}$$

In order to find the maximum margin solution, we solve the following problem:

$$argmax_{\mathbf{w},\mathbf{b}} \frac{1}{||\mathbf{w}||} min_n(t_n(\mathbf{w}^T\phi(\mathbf{x}) + \mathbf{b})$$

With simplification of problem and the soft-margin slack variable , the SVM problem eventually becomes:

$$\begin{aligned} & argmax_{\mathbf{w},\mathbf{b},\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{n=1}^{N}\xi_n \\ & \text{s.t } t_n(\mathbf{w}^T\phi(\mathbf{x}) + \mathbf{b}) \geq 1 - \xi_n; \xi_n \geq 0; n = 1, 2, ..., N \end{aligned}$$

The variable C is the regularization parameter to control the tradeoff between margin and the tolerance of misclassification. In addition,in the dataset is non-linearly separable, we can use kernel trick to transform them to higher linear dimension, such as Gaussian Kernel. Since the SVM problem is a convex optimization problem, we can always obtain a global optimum from the model. With the optimal decision boundary, we can use it to classify our dataset into different labels.

B. Convolutional Neural Networks

With the development in hardware technology, convolutional neural networks have become popular for image classification tasks in the recent years. In convolutional neural networks, the first few hidden layers are convolutional layers. The convolutional layers consist of spatially small filters. These filters are applied through a sliding-window across the input. Each filter outputs an activation map in 2-D that corresponds to an edge, rotation, or some other hidden feature of the input. These activation maps are then aggregated to be the output of that convolutional layer. The key difference of convolutional layers is that they are connected only to a local region of the input volume. Despite of the skepticism CNN received at first,it has been proven by countless papers and against many benchmarks that CNN is actually a very powerful method for image classification. Thus, it was adopted by many researchers in various fields for different tasks.

1) Architecture: Our architecture is similar to the generic Convolutional Neural Network architecture, which usually consists of convolutional layers, followed by pooling layers, normalization layers and finally the fully connected layers. However, we do not use pooling layers. Although they are usually useful and it is counterintuitive to not use them, pooling layer actually decreases the accuracy in our case. Thus we did not use them. During forward propagation, the equation below is calculated at each neuron. Suppose that this neuron has m connections in the previous layer, these connections have weights w_i and the activation values of the

units are a_i.

$$\sum_{i=1}^{m} w_i a_i + b$$

The output of this calculation then goes through a non-linear function. We use ReLu as our non-linear activation function, which clamps the negative activations to 0 and dropout method to provide some relief from overfitting. The ReLu function can be mathematically represented as:

$$f(\mathbf{x}) = max\{0, \mathbf{x}\}$$

Since the gradient of a cost function with respect to its parameters gives the direction of gradient ascent, to minimize the cost function we need to go in the opposite direction. Then, we simply leverage the chain rule and derivatives to update the parameters during backpropagation.

C. Optimizations

One of the problems that should be addressed is the resources neural networks use, since they are a drain on the resources. They are computationally expensive to use. They require a large memory, a powerful battery and a powerful GPU. Therefore, today much of the computation is done via the cloud services. However, recent innovations show that machine learning will become ubiquitious. Edge devices traditionally operate on low resources. Moreover, drones also play an important part in precision agriculture and agricultural technology. We can leverage hyperspectral imaging technology on the drones to provide farmers a real time feedback. Thus, it is important to make neural networks light enough to put them on edge devices.

By optimizating our neural networks, we reduce the number of parameters and computations. Thus, we accelarate inference and decrease the resource requirements. For this reason, we have implemented two methods: (i) layer compression and (ii) layer pruning. Although most of the computation is typically done on convolutional layers, fully connected layers have the most connections and parameters on the convolutional neural networks. Thus, we have implemented these methods on fully connected layers.

1) Fully Connected Layer Compression: Layer compression is used to reduce the number of parameters of a layer. It is usually achieved by decomposing the weight matrix of a layer and then replacing that matrix with a low rank representation or approximation. This method provides a dramatic decrease in the number of parameters.

We replaced only the first fully connected layer by its decomposition, preserving all other layers in the network. First, we used PCA to select the number of principal components that can successfully approximate its weight matrix. Then, we used singular value decomposition to factorize the matrix and replaced this layer with its factorization.

Since each neuron is connected to all neurons in the previous layer, if there are m neurons in the previous layer and n neurons in that layer, then there are mxn connections. Suppose we want to replace the weight matrix $\mathbf{W} \in \mathbb{R}^{mxn}$ with low rank approximation of rank k. The singular value decomposition of $\mathbf{W}$ is below.

$$\mathbf{W} = \mathbf{USV}^T, \ \textit{where } \mathbf{U} \in \mathbb{R}^{m \times m}, \mathbf{S} \in \mathbb{R}^{m \times n} \text{ and } \mathbf{V} \in \mathbb{R}^{n \times n}$$

After selecting the first k principal components of the matrix, we use only $\mathbf{U} \in \mathbb{R}^{mxk}, \mathbf{S} \in \mathbb{R}^{k \times k}$ and $\mathbf{V} \in \mathbb{R}^{k \times n}$. Thus, the matrix $W$is replaced by matrices $\mathbf{R1} \in \mathbb{R}^{m \times k}$ and $\mathbf{R2} \in \mathbb{R}^{k \times n}$.

$$\mathbf{R1} = \mathbf{U}_{m \times k} \mathbf{S}_{k \times k},$$
$$\mathbf{R2} = \mathbf{V}^T_{k \times n},$$

where **R1** and **R2** give the low-rank approximation of the original weight matrix $\mathbf{W}$. We replace the first fully connected layer with two new fully connected layers, constructed from **R1** and **R2**, respectively. While the first replacement layer does not have any bias assigned, the second replacement layer is assigned the bias of the original layer. Thus, we reduce the number of parameters in the matrix from $m \times n$ to $k \times (m+n)$. The change in the number of parameters is dramatic, the results are demonstrated in the results section.

2) Fully Connected Layer Pruning: Layer pruning is another method to optimize the neural networks. The idea is based on reducing the number of connections between the layers. We used a threshold based method, where we masked the connections with respect to the absolute value of their weights. The masked weights are not trained during training can be completely cut-off during deployment with complementary methods.

D. Noise Reduction

The noise reduction was done after the result of SVM and neural networks. As we inspected the resulting image, we found that there were always some dangling misclassifications points. We did the noise reduction by correcting points ,whose neighbors all belonged to another single class, to his neighbor class. We believe that it is impossible for a plant to grow while all the plants near it belonged to another class, and the noise reduction result proved that our belief is correct.

E. Implementation

We used Keras 2.1.5[10] in conjunction with Tensorflow[11] 1.7.0. Keras is an open source maintained by mostly people from Google, where one can quickly prototype neural networks. Nvidia GTX 1070 graphics processing unit was used with CUDA 9.0[12] for calculations. The development was completed on a Windows 10 computer. The modularity of the code was also key aspect of the development.

Other libraries include spectral, matplotlib,[13] numpy,[14] scipy.[15]learn for utilities such as plots, and training-test dataset splitting and image outputs.

We used Adam for gradient descent optimization with a learning rate of 0.0001 and a decay of $1e^{-6}$. Adam acts like a ball with friction that goes down the convex curve of the loss function. To achieve that in addition to storing exponentially decaying average of past squared gradients, it also keeps an exponentially decaying average of gradients and computes an adaptive learning rate for each parameter. Given the 6 GB

RAM of Nvidia GTX 1070, the minibatch size was selected to be 100. This assured the best performance/accuracy for the model.

V. Results

We split our dataset to training and test datasets, %75 and %25, respectively. For CNN, we have included the vanilla classifications (without any optimizations) with PCA-applied and NMF-applied input. With PCA-applied input, our model gives a higher accuracy compared to NMF-applied input. Later, we used the model trained with the PCA-applied data as a baseline and applied our optimizations on top of that model. In this section, we make several comparisons such as retraining and not retraining the model after optimizations. We also demonstrate the results with different ratios of network pruning and different ratios of layer compression. The best accuracies we obtained without denoising are given in the table below.

CNN Vanilla Classification with PCA-Input	CNN with Network Compression	CNN with Layer Pruning
%98.1	%99.1	%98.5

TABLE II: Best Accuracies Obtained using CNN

For SVM, the best accuracies we obtained without denoising are given in the table below.

SVM Gaussian Kernel	SVM Polynomial Kernel
%80.7	%82.7

TABLE III: Best Accuracies Obtained for SVM Kernels

The final accuracies we obtained with best CNN and best SVM after denoising are given in the table below.

CNN with Network Compression	SVM Polynomial Kernel
%99.2	%83.9

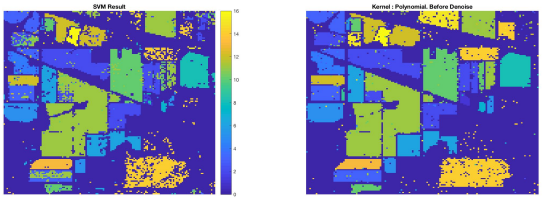
TABLE IV: Final Accuracies after Noise Reduction

A. Support Vector Machines

We provide a detailed explanation of our results, graphs and output images. We have only used Indian Pines dataset as a benchmark. We used %25 of the dataset as the test set and the rest was used as the training set. The operations were completed on the preprocessed data as explained in the preprocessing section.

1) Gaussian Kernel: We start by choosing Gaussian Kernel. The training error is 87.8%, and the testing error is 80.7%. Figure 3 is the resulting classified image. It can be seen that the resulting image successfully shows the structure of the plantation, but there are also many misclassification points. Some classes are well learned but some still need more training.

2) Polynomial Kernel: The best result we can get is from order = 3. The training accuracy for the polynomial kernel is 93.4%, and the testing accuracy is 82.7%. The testing accuracy is better than the Gaussian Kernel. Figure 3 shows the resulting classified image. It can be seen that the Polynomial kernel model has a better result with much less misclassified points.



(a) Gaussian Kernel (b) Polynomial Kernel

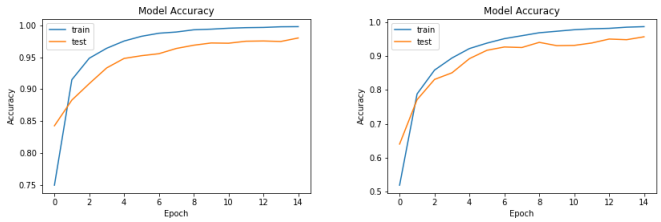
Fig. 3: Support Vector Machine Classification

B. Convolutional Neural Networks

We provide a detailed explanation of our results, graphs and output images. We have only used Indian Pines dataset as a benchmark. We used %25 of the dataset as the test set and the rest was used as the training set. The operations were completed on the preprocessed data as explained in the preprocessing section.

1) Vanilla Classification:

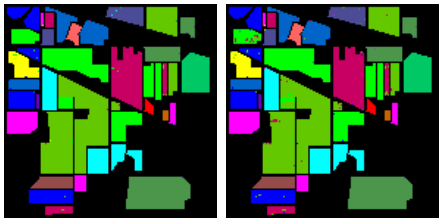
1) PCA
   We implemented two different methods for dimensionality reduction. As explained in the preprocessing section, the input can be successfully approximated by using only the first 35 principal components. The figure 4 shows our results with no optimizations given the first 35 principal components. Learning curve converges around %98.1 accuracy.
2) NMF
   The first 35 components of the nonnegative matrix factorization was used to make a fair comparison with PCA. However, it can be seen from figure 4 below that compared to PCA, NMF produces a poorer result (%95.67 accuracy). This might be due to the fact that NMF does not have negative values, thus each of its components also include very small floating point numbers. During training, this might engender an increased number of errors in calculations due to floating point overflow.



(a) PCA-applied Data Learning Curve (b) NMF-applied Data Learning Curve

Fig. 4: Learning Curves of Vanilla Classifications

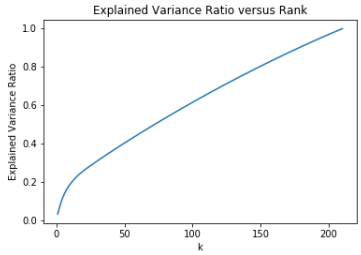
It is clearer from figure 5 that even without any optimizations the classification with the PCA-applied input is better than the NMF-applied input.



(a) PCA Vanilla Classification (b) NMF Vanilla Classification

Fig. 5: Vanilla Classifications

2) Fully Connected Layer Compression: Using the trained model of the PCA vanilla classification, we took the weight of the first fully connected layer and displayed its principal components. The figure below shows that it is nearly linear. Thus, this means that all the dimension of the weight matrix is nearly equally important.

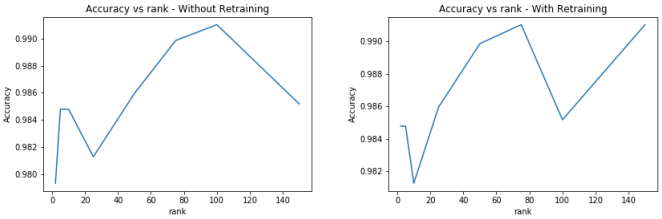


q

Fig. 6: PCA of the First Fully Connected Layer

The reduction in the number of parameters is clearer in the figure below. The first fully connected layer has $m \times n$ parameters. Table VI shows how small it gets with different low-rank representations. Even though we significantly decrease the number of parameters, we get acceptable results.

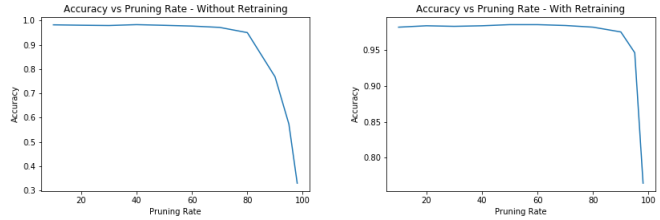
Interestingly, even without retraining there is an increase in accuracy after layer compression. Figure 7 shows that even for very low rank factorizations, we achieve higher accuracy or only a negligible decrease in accuracy. In our case layer compression perturbs the network and prevents overfitting. When k=75, the model peaks at %99.1 accuracy. Moreover, it can be observed that the retraining of the network does little to increase the accuracy. This is because we closely approximate the weight matrix and it is not always necessary to retrain the network after compression.



(a) Without Retraining (b) With Retraining

Fig. 7: Layer Compression: Accuracy vs Rank

3) Fully Connected Layer Pruning: We pruned all the fully connected layers with some ratio k. Only the largest %k of the weights was kept, while the others were clamped to 0. Thus, while the original model had $2430 \times 180 + 180 \times 16$ parameters, this goes down dramatically with pruning. The subsequent reductions are clear in table VI. It can be observed from figure 8 that even without training, this method preserves and slightly increases the accuracy of our model to %98.2, but as k grows the accuracy slowly decreases. However, after %90 of the weights are pruned the accuracy drops significantly. Results are encouraging. Similar to compression pruning works as a complement and prevents overfitting. The accuracy slightly increases as k grows to %60 pruning at %98.5. The difference between retraining and not retraining is clear. While the accuracy drops after %80 percent pruning with the untrained model, the accuracy is relatively preserved until %95 pruning in the retrain model. Moreover, this method is orthogonal to layer compression and can be used in conjunction.



(a) Without Retraining (b) With Retraining

Fig. 8: Uniform Pruning: Accuracy vs Pruning Ratio

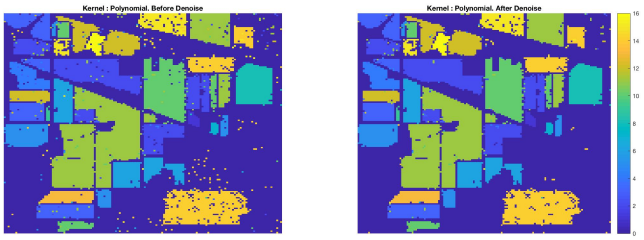
4) Noise Reduction: We run the noise reduction based on our SVM/CNN results. Figure 9 & 10 show the results. In both cases, the noise reduction results are encouraging. For SVM model, the noise reduction provides a bigger boost, there are much fewer dangling incorrect predictions. The accuracy increases from 82.7% to 83.9%. For CNN model, although the original result is very good, there are still some wrong predictions get corrected, and the accuracy increases from 99.1% to 99.2%

Rank of Decomposition	2	5	10	25	50	75	100	150	Original
Number of Parameters	5220	13050	26100	65250	130500	195750	261000	391500	437400
Without Retraining (%)	97.9	98.5	98.5	98.2	98.6	99.0	99.08	98.5	98.1
With Retraining (%)	98.5	98.5	98.1	98.6	98.9	99.1	98.5	99.1	98.1

TABLE V: Layer Compression, Accuracies and Parameter Reduction

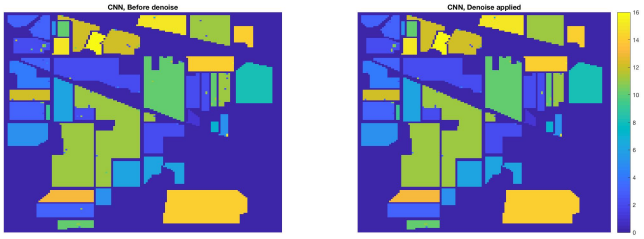
Pruning Ratio (%)	Original	20	30	40	50	60	70	80	90	95	98
Number of Parameters	440280	352224	308196	264168	220140	176112	132084	88056	44028	22014	8805
Without Retraining (%)	98.1	98.2	97.9	98.1	98.2	98.1	95.2	91.4	78.7	55.6	35.4
Without Retraining (%)	98.1	98.2	98.3	98.3	98.5	98.5	98.3	97.7	96.8	95.0	76.2

TABLE VI: Uniform Fully Connected Layer Pruning, Accuracies and Parameter Reduction



(a) SVM before Denoising (b) SVM after Denoising

Fig. 9: SVM Polynomial Kernel result before and after noise reduction



(a) CNN before Denoising (b) CNN after Denoising

Fig. 10: CNN result before and after noise reduction

VI. Conclusion

In this project, we utilized the fact that objects have unique band patterns and proposed machine learning methods of learning the hyperspectral images using SVM and CNN model. We pre-processed our data by oversampling for weak classes, using data augmentation and using PCA to reduce the dimensionality. Then we used SVM and CNN to learn the patterns, and tried optimizations on them. In the SVM model, the result shows that Polynomial kernel with the order of three produces a higher accuracy (83.9%) than the Gaussian kernel. In the Convolutional Neural Networks, optimizations complement training and further increase the accuracy to %99.2. As the result shows, Convolutional Neural Networks performs better than SVM. We think that the reason CNN performs so well is that,CNN is more powerful in fetching structural data. It might be that band patterns themselves have some structure that CNN can interpret, however SVM cannot, thus CNN performs significantly better in this task. The prediction result is very close to the ground truth. We believe that a combination of hyperspectral camera and our CNN model can classify vegetation in a fast, frequent, and cost-efficient manner.

For future works, our model can be extended to plant disease diagnosis through introduction of diseased plants' data. In addition, new data and new classes are always welcome, and they will further enhance the ability of our model. In conclusion, CNN is an effective tool for hyperspectral image analysis and can be utilized in precision agriculture technology.

References

[1] M. F. Baumgardner, L. L. Biehl, and D. A. Landgrebe, *220 band aviris hyperspectral image data set: June 12, 1992 indian pine test site 3*, Sep. 2015. [Online]. Available: `https://purr.purdue.edu/publications/1947/1`.

[2] K. Makantasis, K. Karantzalos, A. Doulamis, and N. Doulamis, "Deep supervised learning for hyperspectral data classification through convolutional neural networks," in *Geoscience and Remote Sensing Symposium (IGARSS), 2015 IEEE International*, IEEE, 2015, pp. 4959–4962.

[3] S. Delalieux, J. Van Aardt, W. Keulemans, E. Schrevens, and P. Coppin, "Detection of biotic stress (venturia inaequalis) in apple trees using hyperspectral data: Non-parametric statistical approaches and physiological implications," *European Journal of Agronomy*, vol. 27, no. 1, pp. 130–143, 2007.

[4] H. Li, W. S. Lee, and K. Wang, "Airborne hyperspectral imaging based citrus greening disease detection using different dimension reduction methods," in *2013 Kansas City, Missouri, July 21-July 24, 2013*, American Society of Agricultural and Biological Engineers, 2013, p. 1.

[5] B. Demir and S. Erturk, *IEEE Xplore*, 2007. [Online]. Available: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4317528`.

[6] G. Mercier and M. Lennon, "Support vector machines for hyperspectral image classification with spectral-based kernels," *IEEE Xplore*, 2018. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1293752.

[7] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1, pp. 37–52, 1987, Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists, ISSN: 0169-7439. DOI: https://doi.org/10.1016/0169-7439(87)80084-9. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0169743987800849.

[8] D. Garber and E. Hazan, "Fast and Simple PCA via Convex Optimization," *ArXiv e-prints*, Sep. 2015. arXiv: 1509.05647 [math.OC].

[9] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[10] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[11] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16, Savannah, GA, USA: USENIX Association, 2016, pp. 265–283, ISBN: 978-1-931971-33-1. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026877.3026899.

[12] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008, ISSN: 1542-7730. DOI: 10.1145/1365490.1365500. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500.

[13] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[14] T. E. Oliphant, *Guide to numpy*, 2nd. USA: CreateSpace Independent Publishing Platform, 2015, ISBN: 151730007X, 9781517300074.

[15] E. Jones, T. Oliphant, P. Peterson, *et al.*, *SciPy: Open source scientific tools for Python*, 2001. [Online]. Available: http://www.scipy.org/.

APPENDIX

A. *Extra Results*

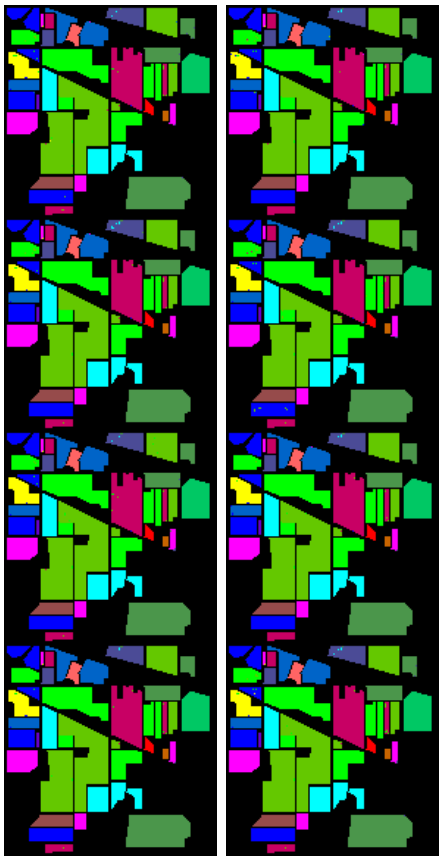


Fig. 11: Layer Compression Before Training (From Low to High Rank Compression)

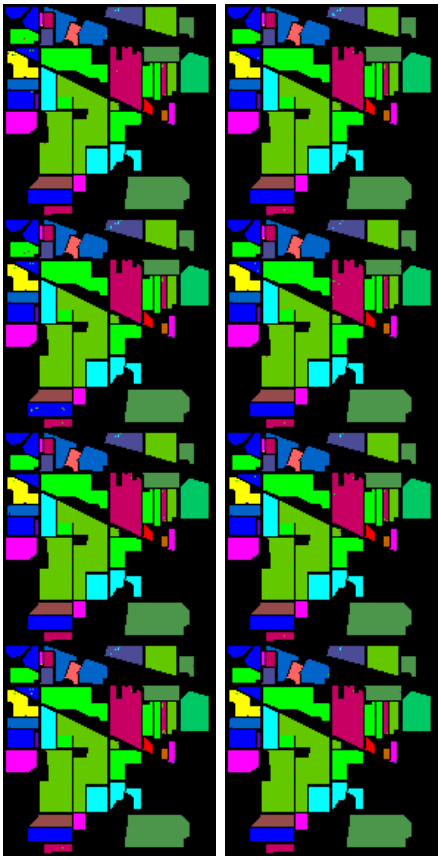


Fig. 12: Layer Compression After Training (From Low to High Rank Compression)

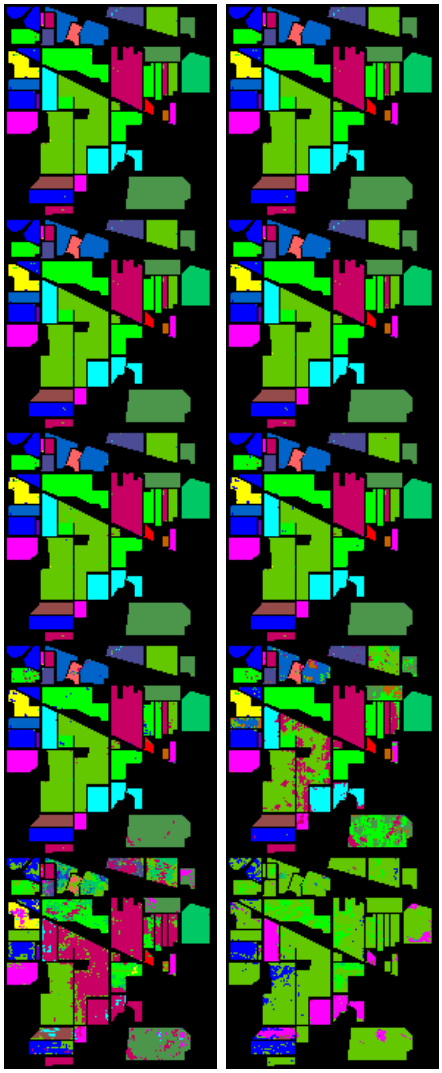


Fig. 13: Layer Pruning Before Training

Fig. 14: Layer Pruning After Training