Fall 2020

Transfer Learning for Hyperspectral Images Utilizing Channel Selection Techniques and Ensemble Methods

Scott Daniel Vogel

San Jose State University

TRANSFER LEARNING FOR HYPERSPECTRAL IMAGES UTILIZING CHANNEL SELECTION TECHNIQUES AND ENSEMBLE METHODS

A Thesis

Presented to

The Faculty of the Department of Electrical Engineering

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Scott Vogel

December 2020

The Designated Thesis Committee Approves the Thesis Titled

TRANSFER LEARNING FOR HYPERSPECTRAL IMAGES UTILIZING CHANNEL SELECTION TECHNIQUES AND ENSEMBLE METHODS

by

Scott Vogel

APPROVED FOR THE DEPARTMENT OF ELECTRICAL ENGINEERING

SAN JOSÉ STATE UNIVERSITY

December 2020

Birsen Sirkeci, Ph.D.	Department of Electrical Engineering
Robert Morelos-Zaragoza, Ph.D.	Department of Electrical Engineering
Guangliang Chen, Ph.D.	Department of Mathematics and Statistics

ABSTRACT

TRANSFER LEARNING FOR HYPERSPECTRAL IMAGES UTILIZING CHANNEL SELECTION TECHNIQUES AND ENSEMBLE METHODS

by Scott Vogel

Hyperspectral images contain information from a wider range of the electromagnetic spectrum than natural images which gives them potential for better classification ability. However, hyperspectral datasets are typically small due to the expensive equipment needed to obtain the images, which can limit classification performance. One solution to this problem is transfer learning, in which a model trained on one dataset is reused for a separate dataset. Research has shown that transfer learning between hyperspectral datasets can give improved performance over models without transfer learning when training data are limited. Since extra hyperspectral data are not always available, the solution proposed here is to instead use networks pretrained on natural image (i.e., red, blue, green, or RGB) datasets for transfer learning. By using various feature selection and feature extraction methods, extracted hyperspectral samples are transformed into a three-channel format to imitate an RGB image and are used for fine tuning the well-known ResNet, DenseNet, and VGG networks. Feature extraction methods include techniques like principal component analysis, which create lower dimensional features from high dimensional spectral data. Alternatively, feature selection methods aim to find the best set of existing channels to use for classification. Experimental results are obtained using two well-known hyperspectral datasets, showing 73.6% accuracy on Pavia University and 82.8% accuracy on Salinas with 25 training samples per class. Additional ensemble methods are implemented that utilize multiple networks and show an increase in accuracy of 4.4% and 3% for Pavia University and Salinas, respectively. These results demonstrate that networks pretrained on RGB datasets are suitable for transfer learning with hyperspectral image datasets and can achieve desirable performance given the proper preprocessing technique.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Sirkeci for her guidance throughout this project. Her knowledge and support made this project enjoyable and allowed me to learn a lot in the process. Her flexibility in working with me also made it possible to accomplish everything in a timely manner. I would also like to thank my two other thesis committee members, Dr. Morelos-Zaragoza and Dr. Chen, for reviewing and verifying the work that I have done.

Finally, I would like to thank my family for their support while writing my thesis and while obtaining my master's degree. Without them, it would have been much more difficult.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN - Convolutional Neural Network

ICA - Independent Component Analysis

ILSVRC - ImageNet Large Scale Visual Recognition Challenge

MMCA - Minimum Misclassification Canonical Analysis

PCA - Principal Component Analysis

RGB - red, blue, green (images)

1 INTRODUCTION

1.1 Motivation

With the advancements in deep learning in the last decade, tasks that have seemed impossible in the past are now achievable. This newly reborn area of machine learning has revolutionized various fields including computer vision, speech recognition, and natural language processing, among others. One prominent example is that as of 2015, neural networks have been able to predict classes of images at a lower error rate than a human, and they have only been improving since [1]. However, even though these networks can accomplish amazing tasks, the complexity of training and developing models can be a barrier to entry for those without the proper access to hardware and large datasets. Although improvements have been made in training speed, efficient training of deep learning models still requires powerful hardware and a graphics processing unit (GPU), which can make large scale deep learning unavailable to those without the proper resources. As recently as 2014, training state of the art deep learning models with multiple GPUs could take as long as 2-3 weeks on large datasets [2]. In addition, access to large datasets needed for deep learning is often limited and gathering enough data for a specific application can be a time consuming and resource heavy task in and of itself. Transfer learning offers a promising solution to these problems.

Transfer learning is a technique that allows a previously trained machine learning model to be used on a new dataset with the assumption that the original model's discriminative abilities will still be useful. Because of this, transfer learning can be one solution to both of the previously defined problems. Since most well-performing models have already been through a thorough training process, the model will likely need less training time to adapt to a new dataset. For the same reasons, less data will be needed to fine-tune the model on a target dataset, so gathering large amounts of data may not be as necessary to achieve high performance. Additionally, since

the deep learning resurgence began, there have been many pretrained machine learning models publicly available online for a variety of applications that have been thoroughly tested and benchmarked, as demonstrated in [1] and [3]. Finding ways to utilize these models to solve new problems can therefore be an efficient way to develop well-performing models.

Hyperspectral imaging is a technology that can offer improvements in many scientific fields including remote sensing, agriculture, and medicine due to its ability to capture detailed information about a scene. Since these images can contain hundreds of spectral channels, they contain much more information than RGB images and thus have the ability to provide better image classification performance. However, one issue with these images is that they are difficult to obtain because of the expensive hyperspectral imaging equipment, which severely limits the availability of large datasets to use for deep learning applications. This makes hyperspectral image data a good candidate for transfer learning, and recent research has shown that transfer learning can improve performance on hyperspectral image classification tasks [4].

1.3 Overview

This work focuses on applying transfer learning principles to hyperspectral imaging datasets. Specifically, well-known pretrained image classification models are applied to these datasets while comparing different feature extraction and feature selection methods. Chapter 2 provides background on hyperspectral images and the pretrained networks utilized in this work. Chapter 3 discusses related literature on hyperspectral imaging that influenced this topic. Chapter 4 will cover the different hyperspectral image datasets used in this work and the techniques used to generate the training and testing data from them. In Chapter 5, the implemented deep learning and preprocessing methods are discussed, and in Chapter 6 the obtained results are analyzed. Finally, Chapter 7 discusses possible future extensions of this work, and Chapter 8 discusses the conclusions that are drawn from it.

2 BACKGROUND

2.1 Hyperspectral Images

Hyperspectral images are cuboids with hundreds of channels in the spectral dimension, unlike RGB images that only contain three spectral channels for red, green, and blue wavelengths. Because of the high dimension of the data, hyperspectral images can capture much more information about a scene than RGB images. In addition, they are not limited to the visual range of the electromagnetic range that humans can see. The Airborne Visible / Infrared Imaging Spectrometer (AVIRIS) developed by NASA's Jet Propulsion Laboratory, for example, captures electromagnet radiation in the range of 400-2500 nm with a spectral resolution of 9.375 nm [5]. In these images, each pixel represents the spectral reflectance of that scene at a specific point throughout a wide range of the electromagnetic spectrum. Fig. 1 shows a sample hyperspectral image and the spectral reflectance curve of three pixels belonging to different classes in the range of 450-850 nm. Since each pixel will represent a different point in the scene it is likely that each pixel will have a different spectral reflectance, although pixels of the same class may share characteristics. This is the main reason that hyperspectral images can accurately capture various characteristics of a scene. While the high dimensionality of hyperspectral images can be a major advantage for image classification, it can also make processing the image a difficult task. It is well known that certain channels of hyperspectral images can contain little information and can be completely discarded [6]. Knowing this, simply using all channels of the image may not offer the best result.

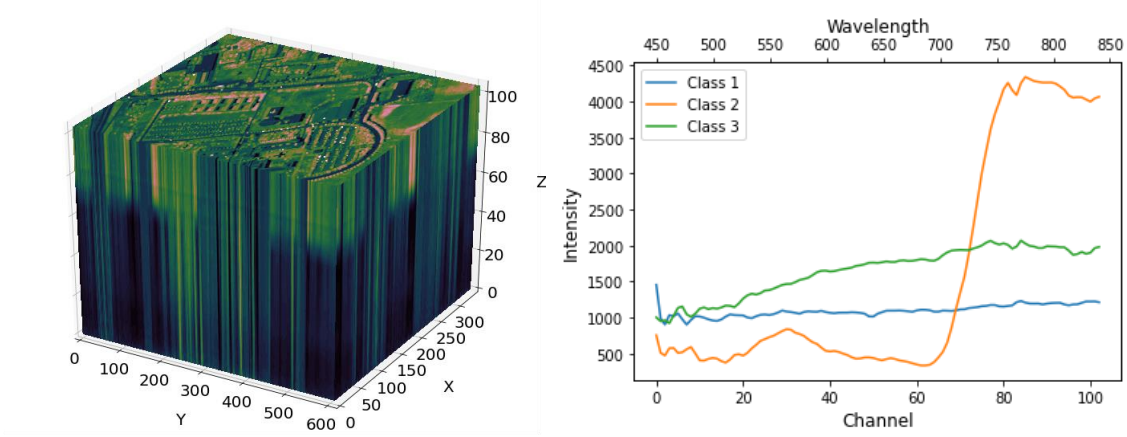

Fig. 1. A sample hyperspectral image (left) and the spectral reflectance of pixels from three different classes (right).

The data-capturing characteristics of hyperspectral images make them suitable for many applications. One of the most popular uses of hyperspectral image datasets is for remote sensing. In [7], researchers were able to classify different species of native and non-native trees using aerial-view hyperspectral images with mean F1-scores ranging from 76%-89%. By using hyperspectral images of vineyards, researchers in [8] were able to identify areas of water stress with 83% accuracy, which in turn helped to improve the crop quality and sustainability of vineyards by preventing crop loss. Apart from remote sensing, another field that has utilized hyperspectral imagery is medicine. Medical applications include cancer detection and disease diagnosis, where abnormal cancer tissues or disease symptoms show altered spectral characteristics that help to improve classification [9]. This wide range of use cases for hyperspectral images makes them a useful tool that will benefit many different scientific fields.

2.2 Transfer Learning and Pretrained Networks

With the rapid progression of deep learning in recent years, one of the largest areas of improvement has been in computer vision. The optimization of the convolutional neural network, or CNN, was one of the main drivers of these improvements. In 2010, an annual computer vision

challenge called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) began that tested competitors on their ability to correctly classify images in a large dataset containing thousands of classes and millions of total images [10]. From 2012-2015 alone, the top-5 classification error rate on ImageNet, which is the rate at which the target class was not one of the top five classes predicted by the network, dropped from 16.42% [1] to 3.47 % [11]. Advancements in neural network architecture were key to this improvement in performance, and many architectures have become famous since. The architectures used in this paper are listed in Table 1 and are further described below.

Table 1

Pretrained Models Used in This Paper

Model Architecture	VGG	ResNet / ResNetV2	DenseNet
Fewest Layers	VGG-16	ResNet50 / ResNet50V2	DenseNet121
_	_	ResNet101 / ResNet101V2	DenseNet169
Most layers	VGG-19	ResNet151 / ResNet151V2	DenseNet201

The VGG architecture is one of the first network architectures that was able to include a very deep set of sequential convolutional layers. The main contribution in [2] is the implementation of CNNs with 16-19 weight layers while using a small convolutional filter size of 3×3. This small convolutional filter size was a key change that allowed deeper networks to be properly trained and showed that stacking many layers with a small filter size could perform better than fewer layers with larger filter size. This was because of the realization that stacking multiple layers of 3×3 filters can produce the same result as a larger filter size while also including additional non-linear activation functions [2]. Using these methods, the VGG networks were able to accomplish a top-5 error rate of 7.32% on the ImageNet challenge, a 4.4% improvement over the best network from the previous year (2013) [1]. An example of this design architecture for VGG-19 is shown below in Fig. 2.

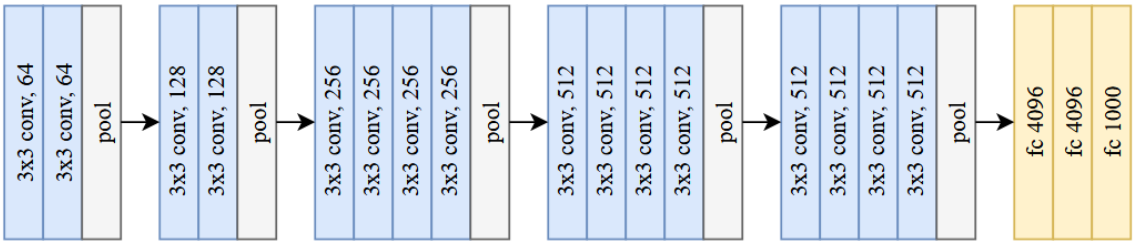

Fig. 2. The VGG-19 network architecture. Here, '3×3 conv, 64', refers to a convolutional layer with 64 3×3 filters, 'pool' refers to a pooling layer that reduces the size of the mapping by 1/2, and 'fc 4096' refers to a fully-connected layer with 4096 neurons.

Building upon the success of the VGG networks in [2], a new network architecture named ResNet was developed that included a new structure called the residual unit and an extension of that unit called the bottleneck unit [11]. The addition of these units allowed networks of many more layers (as many as 1000 layers were tested) to be trained properly by reducing the effect of the vanishing gradient problem [11]. An example of the two-layer residual unit is shown on the left in Fig. 3, containing two convolutional layers with 3×3 filters. Essentially, the residual unit takes the output of a layer and not only sends it through the following layer but also adds it to the output of the following layer without modification. The bottleneck unit uses this same idea but creates a three-layer convolutional block with 1×1, 3×3, and 1×1 filters instead of two 3×3 filter convolutional layers. ResNets designed with this bottleneck structure are regularly referred to as ResNetV2. These architectures were able to achieve additional improvements on the ImageNet data set with a 3.57% top-5 error rate, a 3.75% improvement over VGG-19 [11].

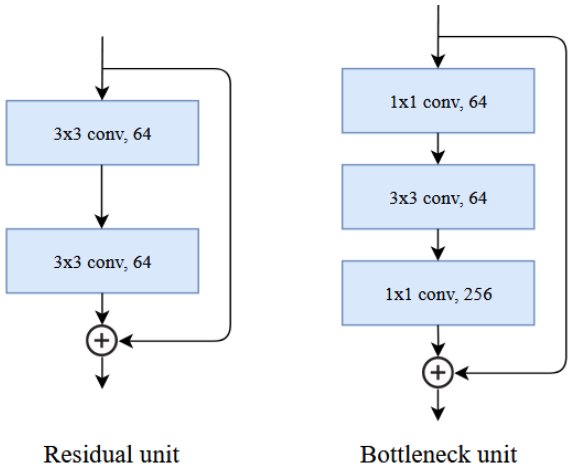

Fig. 3. The residual unit (left) and bottleneck unit (right). As in Fig. 2, ‘conv 3×3, 64’ here refers to a convolution layer with 64 3×3 filters.

The third network architecture utilized in this work is the DenseNet architecture that was first developed in 2017 [12]. This architecture takes inspiration from ResNet and develops the idea of densely connected blocks where each layer takes the output from every previous layer in the block. This structure is visualized in Fig. 4, where each block of feature maps is connected to the next and the convolution operation is skipped for those features. The output of every convolutional layer is added to the input of every layer, except for the first and last layers that represent the start and end of the five-layer block. With this technique, researchers were able to achieve a top-5 error rate of only 5.54% [12].

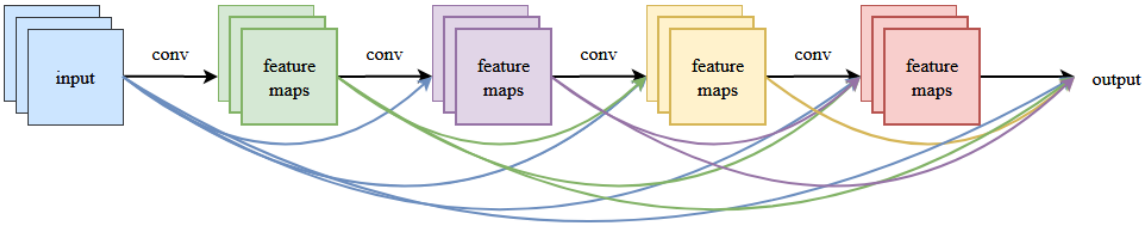

Fig. 4. The ‘dense block’ structure of DenseNet.

While these models have shown great performance, the training process is difficult without the proper resources. Transfer learning is a growing area of research in the field of machine learning and more specifically, in the field of deep learning, that can allow these networks to be reused for different applications. Through transfer learning, model development time greatly decreases as does the need for very large datasets. The basic concept behind transfer learning with convolutional neural networks is that networks will tend to learn generic features at the beginning of the model and learn increasingly specific features towards the end. The first convolutional layers extract high level features maps that slowly become more specific after each convolutional layer. Because of this, the convolutional layers can be seen as generic feature extractors that can work for a variety of data that share similar structure. In the final fully connected layers, the network combines the newly created features in a way that lets it discriminate between different classes. Knowing this, one can re-use the convolutional layers of a CNN to extract features and simply re-structure the fully connected layers and train them on the target dataset. For small datasets, this can work well since most of the network's weights have already been learned on a much larger dataset. This can apply directly to hyperspectral datasets, which suffer from small sample size. However, significant adaption is needed to utilize pretrained CNNs with these datasets.

In order to use these pretrained networks with hyperspectral images, either the hyperspectral image or the network must be adapted for the application. Since nearly all available pretrained models for image classification are on developed RGB image datasets with three spectral channels, one of the main issues to solve is dealing with the high dimensionality of the hyperspectral image data. Some method must be chosen to convert the hyperspectral image samples into three channel versions, whether that is done by selecting a set of channels or creating a new set of channels with a transformation. Additionally, hyperspectral samples are

typically treated as single pixels from the image, and that is not suitable for a network that takes in samples with a rectangular shape. This means that other adaptions must occur in the spatial dimensions. Alternatively, a network could be adjusted to allow for samples with higher dimension or single pixel inputs by changing the input channels of the network. Although there are potentially many ways of accomplishing this, the chosen technique will likely have a significant effect on the model performance, which is makes it an opportunity for exploration.

3 RELATED WORK

Applying deep learning models to hyperspectral image classification has been a popular research area in recent years. CNNs have been particularly popular for hyperspectral image classification, as seen in [11], [12], and [2], due to their success in various computer vision tasks and contests like the previously mentioned ILSVRC [1]. Researchers have applied CNNs in many different ways in order to solve the hyperspectral image classification problem. In [13], researchers utilized the ResNet architecture and adapted it to be capable of 3-D convolution by simply inflating the network's weights to 3-D. This 3-D convolution allows both spectral and spatial features to be learned, unlike typical CNNs that only perform convolution in the spatial dimensions. With the high dimensionality of hyperspectral data in the spectral dimension, including spectral features intuitively makes sense. Using this technique, researchers were able to show significant improvements in classification accuracy (12-17%) over similar methods that only implemented 2-D convolution for the Pavia University dataset [13].

Recent research has also shown that there can be improvements in classification performance for hyperspectral datasets by pre-training networks on not only different hyperspectral datasets, but also on RGB image datasets. In [4], researchers proposed two methods of transfer learning using their custom neural network architecture that accepts 3D samples extracted from the hyperspectral image. In the first method, a 3-D CNN model is first trained on one hyperspectral dataset, and the final fully connected layers are fine-tuned on a target HSI dataset with a small number of samples per class (25 and 50). Using this approach, results showed improvements in the range of approximately 1.19-4.17% over networks without transfer learning when there were 25 samples per class and improvements of 0.05-2.68% when there were 50 samples per class [4]. This implies that transfer learning may show more significant improvements when lower numbers of samples are available for training on the target dataset, which is consistent with the goal of

transfer learning. In the second approach, RGB datasets were used as the source for transfer learning with a target hyperspectral dataset. In order for the RGB datasets to work with their 3D-CNN architecture, RGB images were repeated along the third dimension so that the number of dimensions would be equal to that of the target hyperspectral dataset. With this approach, results showed improvements of 1.86-8.58% over the network without transfer learning when there were 25 samples per class and improvements of -0.07-2.75% when there were 50 samples per class [4]. These results suggest that transfer learning for hyperspectral datasets can be beneficial with varied sources of data for pre-training.

Another transfer learning approach in [14] utilized a popular pretrained network, VGG-16 [2], to work as a part of its deep learning system. Their methods also included multiple different RGB datasets to use for training different parts of the network. In this approach, the VGG-16 network is first trained on the ImageNet dataset for which it was originally developed [2]. Then, the VGG-16 network is adapted to pixelwise prediction by including multiple new deconvolution and upsampling layers to reformat the shape of the hyperspectral input data. Third, this adjusted network is fine-tuned on the PASCAL VOC 2011 datasets, a dataset that includes pixel-level image segmentation [15], to learn additional pixel-level features that may not be learned in the hyperspectral dataset. Finally, this pretrained network is used to extract features from the hyperspectral data to use for training the final prediction layers of the network. Using these techniques for pretraining, researchers were able match the performance of other well-performing models that did not use any transfer learning techniques, which further amplifies the conclusion that transfer learning can be useful for hyperspectral image datasets.

While many deep learning based approaches for hyperspectral image classification use all channels of the hyperspectral image, other research has aimed to find the best set of channels to use, since it is likely that some channels will offer poor classification performance. In [16],

researchers describe six categories of channels selection methods which are ranking-based, searching-based, clustering-based, sparsity-based, embedding-learning based, and hybrid scheme-based methods. While these methods differ in implementation, they all aim to find the best set of channels to use for classification. The most intuitive methods may be the ranking-based methods, in which channels are ranked by their estimated classification ability. Channels can be ranked by many different metrics, including high information criteria or low correlation criteria, to determine the channels that are best suited for classification [16]. Once channels are ranked, a set of decorrelated channels can be chosen to give varied sources of data. In clustering methods, channels are clustered by their similarity, as defined by the algorithm, and from each cluster a representative channel is chosen that shares the characteristics of the cluster. In this way, representative channels from separate clusters should be dissimilar and will be likely to hold different information. By reducing the number of channels, these techniques can help to alleviate the problem of dimensionality mismatch between hyperspectral images and typical CNN networks.

4 DATASET DESCRIPTION

4.1 Pavia University and Salinas Hyperspectral Datasets

The two datasets used in this paper are the Pavia University and Salinas scenes which are publicly available online and are widely used in research [6]. Both scenes are aerial views of ground-level objects including classes like asphalt, meadows, and celery, with nine classes for Pavia University and sixteen classes for Salinas. For these datasets, samples are organized as single pixels that cover the entire spectral range of the image. This means that for a hyperspectral image with shape H×W×L, a single pixel sample will have the shape 1×1×L. One thing to note is that although each pixel belongs to a single class, a large number of the pixels in these datasets (79% for Pavia University and 51% for Salinas) are labeled as class zero, which identifies them as background. Following the practices shown in [14], [17], and [13], the background class is not included as one of the classes to identify. These datasets are captured by different sensors and therefore have different spatial and spectral resolutions, which makes them interesting to compare, since this difference may result in differences in classification performance.

Fig. 5 below shows the ground truth and two sample channels from the Pavia University dataset, which has dimensions 610×340×103 channels. The ROSIS sensor captured the Pavia University scene and offers spatial and spectral resolutions of 1.3 meters and 4.0 nm over a spectral range of 430 to 860 nm [18]. Each color in the ground truth represents a different class, with the color black representing background (class 0). As seen with the two sample channels, the characteristics of the data can vary significantly between channels, with some classes being more distinguishable in channel 96 than in channel 1. This is the key motivation for utilizing feature selection and feature extraction methods.

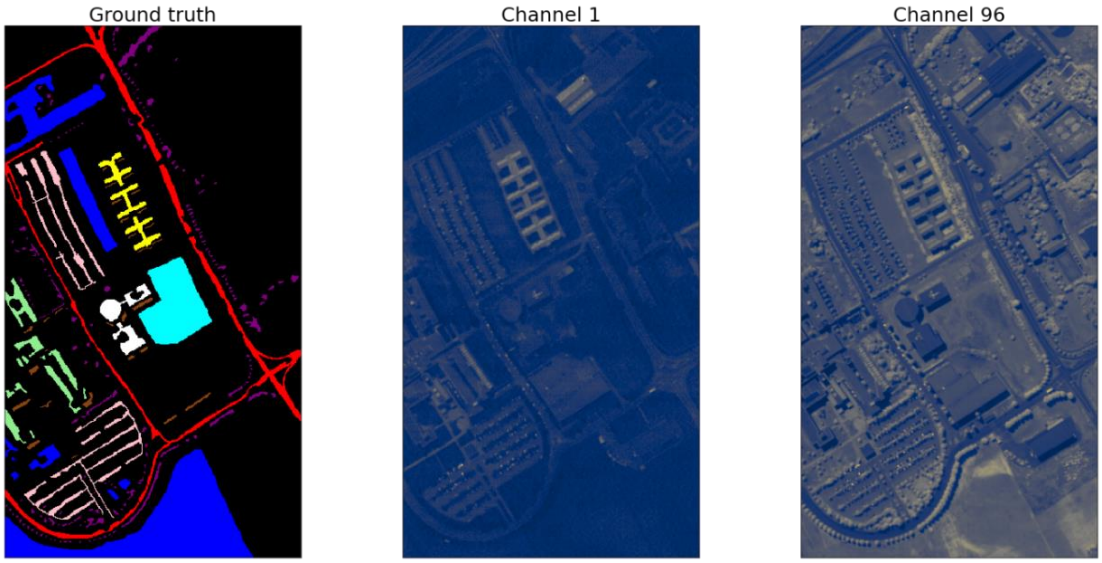

Fig. 5. The ground truth and two samples channels from the Pavia University dataset.

The Salinas scene was captured by the AVIRIS sensor at NASA's Jet Propulsion Laboratory and offers spatial and spectral resolutions of 3.7 meters and 9.375 nm over a spectral range of 400 to 2500 nm, well beyond the range of the visible light spectrum [5]. The original shape of this hyperspectral image is 512×217×224 channels; however, 20 of the channels (108-112, 154-167, and 224) provide little information due to water absorption and are removed to result in a 512×217×204 channel image [6]. As shown in the Pavia University dataset, the two sample channels have much different characteristics. Channel 1 of the Salinas dataset is a very noisy representation of the scene with hard to distinguish classes, while channel 190 shows a visible distinction between most classes.

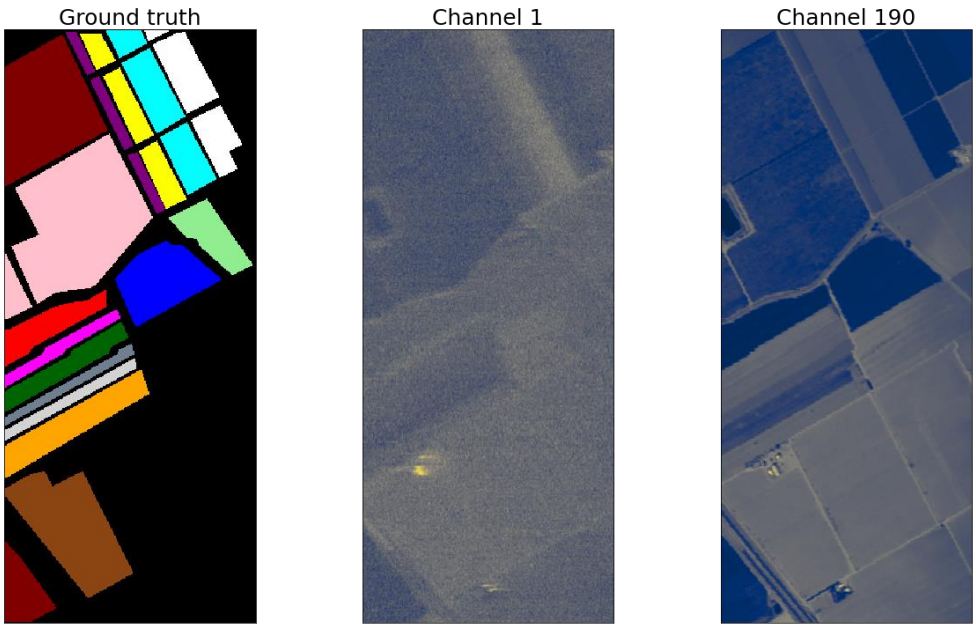

Fig. 6. The ground truth and two samples channels from the Salinas dataset.

4.2 Dataset Generation

Because the pretrained networks used here expect a 3-channel image with height and width of at least 32×32, samples cannot be taken as single pixels and must be transformed into a suitable shape [3]. One method proposed by [4] is to extract samples as N×N×L (where L is the number of channels) cuboids from a sliding window across the image's first two dimensions and select the center pixel's class to be the chosen class. As mentioned previously, any samples with class 0 are labeled as background, so if a sample's center pixel is class 0 then the sample is discarded. This creates samples large enough to feed to the network and provides additional context information to the samples, since nearby pixels are included that will likely have information relevant to the target class. The authors in [4] used a 27×27 window to extract samples that could be fed into a custom neural network architecture. Following these practices, a 33×33 (odd to allow a center pixel to exist) window was originally chosen to extract samples from the raw data. However, using such a large window size has the potential to create a bias problem where too much of the

information in a given training sample is also available in the test samples. For example, in one 33×33×L pixel sample and another shifted by one pixel, approximately 94% of the pixels are shared which will greatly bias the training data. To bypass this issue, the window's size was decreased to 5×5 and samples were taken every 5 pixels so that no pixels were repeated in any samples. After using a feature selection or feature extraction method that results in M channels, these samples are interpolated per channel so that they fit the target shape of 33×33 in the first two dimensions. The interpolation is accomplished with a cubic spline using SciPy's zoom function in the 'ndimage' module [19]. This process of converting a single 5×5×L to a 33×33×M sample is visualized below in Fig. 7. The choice of M here depends on the feature preprocessing method used which is further explained in Chapter 5

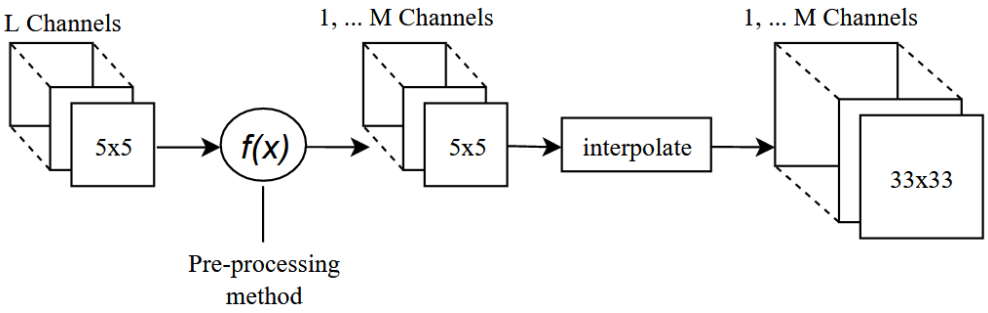

Fig. 7. The process of converting 5×5×L shaped samples to 33×33×M.

While this method decreases the number of samples available for training and test data, it provides a more accurate representation of transfer learning, since the data are not heavily biased. Table 2 shows the class distribution for the Pavia University dataset using the 5×5 sliding window approach. For the training set, 25 samples were randomly chosen per class, except in the case where there were less than 50 total samples in the class (e.g., class 9 – Shadows). It is noteworthy that the dataset is highly unbalanced with 42% of the samples belonging to class 2 –

Meadows and only 2.3% of samples belonging to the smallest class, class 9 – Shadows. Approximately 17% of the data are used for training and validation with the rest used for testing.

Table 2

Class Distribution for the Pavia University Dataset

Class	Training	Validation	Test
1 - Asphalt	25	8	232
2 - Meadows	25	8	721
3 - Gravel	25	8	54
4 - Trees	25	8	85
5 - Painted metal sheets	25	8	19
6 - Bare soil	25	8	169
7 - Bitumen	24	8	16
8 - Self-Blocking Bricks	25	8	109
9 - Shadows	19	8	11
Total	218	72	1,416

Table 3 below shows the distribution for the Salinas dataset. The training and validation sets are distributed in the same manner as Pavia University, with 25 and 8 samples per class, respectively. As with the Pavia University dataset, the classes are not balanced; however, the Salinas dataset does not have a single class that dominates as in the Pavia University dataset but rather has many classes with very few samples. In total, approximately 29.5% of the total data are used for training and validation.

Table 3

Class Distribution for the Salinas Dataset

Class	Training	Validation	Test
1 - Broccoli_green_weeds1	25	8	49
2 - Broccoli_green_weeds_2	25	8	115
3 - Fallow	25	8	48
4 - Fallow_rough_plow	25	8	22
5 - Fallow_smooth	25	8	70
6 - Stubble	25	8	120
7 - Celery	25	8	111
8 - Grapes_untrained	25	8	149
9 - Soil_vineyard_develop	25	8	219
10 - Corn_senesced_green_weeds	25	8	98
11 - Lettuce_romaine_4wk	22	8	13
12 - Lettuce_romaine_5wk	25	8	44
13 - Lettuce_romaine_6wk	18	8	10
14 - Lettuce_romaine_7wk	22	8	13
15 - Vineyard_untrained	25	8	112
16 - Vineyard_vertical_trellis	25	8	38
Total	387	128	1231

In order to simulate datasets of varying complexity, each of the datasets are re-created with subsets of the data containing from 2-C classes, where C is the total number of classes in the dataset. For example, a three-class version of the Pavia University dataset might consider only those samples that belong to classes 1, 2, and 3. The class subsets for each dataset were chosen randomly and remain the same throughout all experiments to provide repeatability. The different classes included in each subset are shown in Table 4

Table 4

Data Subsets for the Salinas and Pavia University datasets.

Number of classes	Salinas	Pavia University
2	9, 15	4, 8
3	2, 9, 15	3, 4, 8
4	2, 9, 10, 15	3, 4, 6, 8
5	2, 9, 10, 12, 15	3, 4, 6, 7, 8
6	1, 2, 9, 10, 12, 15	3, 4, 6, 7, 8, 9
7	1, 2, 9, 10, 12, 15, 16	1, 3, 4, 6, 7, 8, 9
8	1, 2, 3, 9, 10, 12, 15, 16	1, 3, 4, 5, 6, 7, 8, 9
9	1, 2, 3, 5, 9, 10, 12, 15, 16	All 9 classes
10	1, 2, 3, 5, 7, 9, 10, 12, 15, 16	-
11	1, 2, 3, 5, 7, 9, 10, 11, 12, 15, 16	-
12	1, 2, 3, 5, 7, 9, 10, 11, 12, 14, 15, 16	-
13	1, 2, 3, 5, 6, 7, 9, 10, 11, 12, 14, 15, 16	-
14	1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16	-
15	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16	-
16	All 16 classes	-

5 IMPLEMENTED METHODS

5.1 Model Implementation

All models are trained using the fine-tuning method. The goal of fine-tuning is to utilize the inner weights of the network that have already been learned through training on a separate dataset and only train new weights for the final fully connected layers. For each pretrained network, as listed in Table 1, the final fully connected layers of the network are removed and replaced with a 2-D global average pooling layer followed by a fully connected layer that predicts the class of the data. This results in very few weights to learn, as only the fully connected layer has parameters to adjust. Essentially, the pretrained network, up to the final fully connected layer, is only used to extract features from the data and is used as a complex data preprocessing step. The output from the pretrained network is the new data used to train the fully connected output layer.

As discussed previously in Chapter 4, the samples generated from the original hyperspectral data are of shape 5×5×L before being transformed into a 5×5×M shape and finally a 33×33×M shape. The choice of M here determines the number of channels that are used in the model. In the case where M = 1, the single channel is repeated three times and fed through the pretrained network, and the fully connected output layer is trained on this output. When M > 1, an ensemble technique is used to generate the output, as visualized in Fig. 8. As in the case where M = 1, each of the M channels are repeated three times before going through the network, but here, one separate fully connected layer is trained for each of the M channels. With this technique, predictions come as an ensemble, and the most commonly predicted class is chosen. That is, with M channels there will be M unique predictions for each sample. In addition to this ensemble technique, an ensemble of the 11 pretrained networks in Table 1 is also implemented to offer further improvements in performance. In this ensemble, the models mentioned previously are

developed using all of the 11 pretrained models, and the most common prediction among these 11 models is determined to be the predicted class.

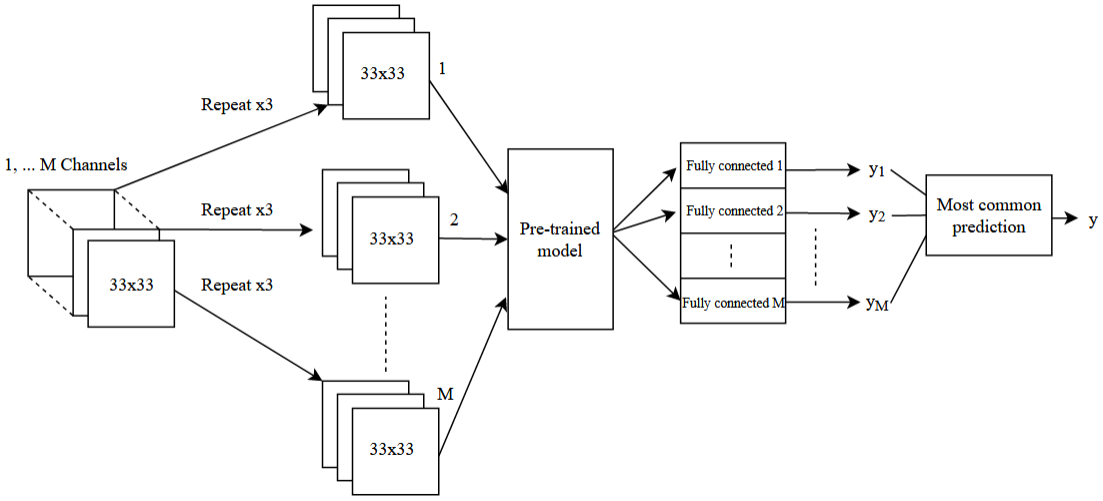

Fig. 8. Visualization of the channel ensemble process.

5.2 Channel Preprocessing Methods

5.2.1 Greyscale

The simplest preprocessing method used here is a greyscale technique. This is done by converting the hyperspectral sample to a single channel by averaging all of the channels and repeating the single channel three times.

$$C_{greyscale} = \frac{1}{L}\sum_{l=1}^{L} C_l \tag{1}$$

Since this method is simple, it is likely that the more complex and intelligent methods will outperform it. For this reason, it is used as a baseline to compare against the performance of other models. If more complex methods do not show improvements, then they are not suited for this application.

5.2.2 RGB

The RGB method is implemented by selecting channels from the image that correspond to red, green, and blue wavelengths. The idea here is to create an actual RGB image using the current hyperspectral data, since the pretrained networks were originally trained on RGB images. One important thing to consider about this method is that not all hyperspectral images contain red, green, and blue wavelengths, which means that this method is not always possible. For this reason, the RGB method will also be used as a reference to compare with other methods. Since Pavia University and Salinas both contain RGB channels, the channels with wavelengths nearest to the RGB wavelengths are chosen for those channels. The wavelengths used to represent red, green, and blue are 682.5 nm, 532.5 nm and 467.5 nm, respectively. Table 5 shows the chosen channels for each of the datasets. Since the spectral range and number of channels are known, they can be used to calculate the wavelengths for each channel, and the channel nearest to the target wavelength is chosen. Unlike all other preprocessing methods, each of the three channels are not repeated three times here, and the 3-channel images are passed directly into the network.

Table 5

RGB Channels for Pavia University and Salinas

Dataset	Spectral Range	Number of Channels	Red	Green	Blue
Pavia University	430-860 nm	103	60	24	9
Salinas	400-2500 nm	224	30	14	7

5.2.3 K-Means Clustering

K-means clustering is a method used to cluster data points that are close in distance to each other. The measure of distance used here is the Euclidean distance between data points:

$$d(b,c) = \sqrt{\sum_{x=1}^{W}\sum_{y=1}^{H}\left(b_{x,y} - c_{x,y}\right)^2} \tag{2}$$

Here, b and c are the channels of the hyperspectral image, and x and y vary over the height (W) and width (H) of the channel. Using this distance, each sample is assigned to a nearby cluster of data points in 1-K different clusters. As described in [16], the algorithm uses the channels as samples so that each channel can be grouped with other similar channels. Two different methods are implemented with this algorithm using the KMeans module in Scikit-learn's cluster package [20]. The first method is to generate K clusters with all of the bands and average the bands in each of the clusters to obtain a single channel for each cluster, as shown in (2). Here, k is the number of the cluster, $|k|$ is the number of channels in cluster k, c_l is a channel in cluster k, and C_k is the average of all channels in cluster k.

$$C_k = \frac{1}{|k|} \sum_{C \in k} C \tag{3}$$

These cluster averages are then used as the inputs for training K output networks as shown in Fig. 8. In this implementation, the choice of K here is set to 3, so that there are three final channels to train three different fully connected layers. This method can be considered analogous to a three-channel version of the greyscale method and will demonstrate how well a simple ensemble of three channels will perform.

The second method implemented with K-means clustering is similar to the first, but it does not average any channels and can be considered a feature selection method. In this method the clusters are generated the same way with the K-means clustering algorithm, but the bands closest to the centroid of the cluster are chosen as representative bands for the cluster as demonstrated in equation (4). Essentially, the Euclidean distance (d) between each channel and its corresponding cluster centroid ($Centroid_k$) are computed, and the channel with the smallest distance to the centroid is chosen to represent the cluster (I_k). The centroid here is the average of all channels in

the cluster. After K channels are chosen, they are used for an ensemble network as described previously. This method is implemented with K =3 and K=5.

$$I_k = argmin(D),\ \ D = \{\, C \in k \mid d(C, Centroid_k) \,\} \tag{4}$$

5.2.4 Minimum Misclassification Canonical Analysis

Minimum misclassification canonical analysis (MMCA) is a technique that ranks the image's channels by their classification ability and originates from Fisher's discriminant function [21]. The algorithm as described in [21] begins by first computing m, the mean of all samples and m_i, the mean of each class i:

$$m = \frac{1}{N}\sum_{i=1}^{C}\sum_{j=1}^{N} x_{ij} \tag{5}$$

$$m_i = \frac{1}{N_i}\sum_{j=1}^{N_i} x_{ij} \tag{6}$$

Here, C is the number of classes, N is the number of samples, N_i is the number of samples in class i, and $x_{ij}$is the j_{th} sample in class i. These values are used to form the total, between-class, and within-class scatter matrices, S_T, S_W, and S_B:

$$S_T = \frac{1}{N}\sum_{i=1}^{C}\sum_{j=1}^{Ni}\left(x_{ij} - m\right)\left(x_{ij} - m\right)^T \tag{7}$$

$$S_W = \frac{1}{N}\sum_{i=1}^{C}\sum_{x_{ij}\in\omega_i}\frac{1}{N}\left(x_{ij} - m_i\right)\left(x_{ij} - m_i\right)^T \tag{8}$$

$$S_B = \sum_{i=1}^{C}\frac{N_i}{N}(m_i - m)(m_i - m)^T \tag{9}$$

With these scatter matrices, the following eigenvalue problem in (10) can be solved. Using the resulting eigenvalues and eigenvectors, r_{ik} (11) can be used to calculate p_k (12), which can be viewed as the variance or classification ability of band k. This means that ranking bands by p_k (descending) will order them according to their classification ability.

$$S_B v_i = \lambda_i S_W v_i \tag{10}$$

$$r_{ik} = \sqrt{\lambda_i} v_{ik} \tag{11}$$

$$p_k = \sum_{k=1}^{l} r_{ik}^2 \tag{12}$$

After the channels are ranked, they are decorrelated to find a set of M channels using a correlation measure based on Kullback–Leibler divergence and an algorithm inspired by the one used in [22]. Here, $D(p, q)$ is the total divergence between bands and $L(p, q)$ is the divergence from channel p to channel q, while p and q are the grey-level histograms of each band. In the implementation here, the values contained in p and q are calculated such that they are distributed amongst 512 discrete bins.

$$D(p, q) = L(p, q) + L(q, p) \tag{13}$$

$$L(p, q) = \sum_{i=1}^{l} p_i log \frac{p_i}{q_i} \tag{14}$$

The algorithm for finding the best set of M decorrelated channels is a modification of the algorithm described in [21]. It begins with the top ranked channel and sequentially adds channels to a set that are greater than a certain divergence threshold when compared to every channel in the current set. At the start, the set is empty, so the highest ranked channel is always a part of the set. After all the channels have been compared each other, if the number of channels that satisfy

the criteria is greater than M, the divergence threshold must be increased, and the process must be repeated until the number of remaining bands is equal to M. If the number of chosen channels is less than M, the threshold must be decreased. In the case that there cannot be only M remaining channels, M is increased by 1, and the algorithm is repeated until the number of remaining channels is equal to the new M, and only the original best M bands are returned. Algorithm 1 below summarized this process. By decorrelating this way, channels with the best classification ability are prioritized while also ensuring that channels in the set have low correlation.

Algorithm 1: Channel Decorrelation for MMCA

1. Start with the set of ranked channels S_{ranked} from MMCA and desired number of channels M
2. Set a divergence threshold d_c, and a minimum divergence d_{min}, $M_{current} = M$
3. Initialize an empty set of channels S_{out}
4. Compute $D(C_i, C_j)$ for channel C_i in S_{ranked}, with every channel C_j in set S_{out}
5. If $D(C_i, C_j) < d_c$ for every C_j in S, add C_i to the set S_{out}, else continue to channel C_{i+1}
6. After all channels in S_{ranked} have been compared to the set S_{out}, if:
   (a) $|S| = M_{current}$, return M top channels from S
   (b) $|S| > M_{current}$, increase d_c by 50% and start from step 3
   (c) $|S| < M_{current}$ and $d_c > d_{min}$, decrease d_c by 50% and go to step 3
   (d) $|S| < M_{current}$ and $d_c < d_{min}$, increase $M_{current}$ by 1 and go to step 3

5.2.5 Principal Component Analysis

Principal component analysis (PCA) is a feature extraction and dimension reduction technique used to emphasize areas with high variance and likely high levels of information [23]. With PCA, the number of components is a parameter that is chosen to suit the application, and the number of components can range from 1-L, where L is the total number of features (channels). The principal components follow a set of criteria when they are created that gives them a specific

order. The first principal component is found along the axis that contains the highest variance in the features, and thus can contain a large amount of information. The second principal component is along the axis that contains the highest variance and is also orthogonal to the first principal component. For the third principal component, it is along the axis that contains the highest variance and is also orthogonal to the first two principal components, and this pattern continues for all subsequent components [23]. This also implies that the amount of information present in each principal component will decrease as the principal component number increases. Additionally, since each of the principal components are orthogonal to each other the information shared between them is minimal which will likely make them favorable for an ensemble.

Calculating the principal components entails optimizing a linear transformation of the input features, where the input features are the channels of the hyperspectral image samples. Equation (15) shows the formula for the principal component Z_l, the l_{th} principal component, where X_p is the p_{th} feature and φ_{pl} is the loading factor for the p_{th} feature of principal component l. The loading factors are found by maximizing the sample variance while also satisfying the constraint in (16) and satisfying the orthogonality criteria mentioned previously [23].

$$Z_l = \varphi_{1l}X_1 + \varphi_{2l}X_2 + \cdots + \varphi_{pl}X_p \tag{15}$$

$$\sum_{j=1}^{p} \varphi_{jl}^2 = 1 \tag{16}$$

$$\varphi_l = \left(\varphi_{1l}, \varphi_{2l}, \ldots \varphi_{pl}\right)^T \tag{17}$$

Once the loading vector φ_l for principal component l is calculated using the training data the transformation is applied to the validation and test data as well. This results in an image with M channels, where M is the number of principal components and each channel is a principal

component of the data. The implementation here uses the IncrementalPCA module from Scikit-learn, which allows the PCA loading vectors to be calculated incrementally to save on memory usage [24]. Finally, before calculating the principal components the data are also centered as a preprocessing step.

5.2.6 Independent Component Analysis

Independent component analysis (ICA) is another feature extraction and dimension reduction method that shares some similarities to PCA. ICA aims to transform set of signals into several subcomponents that are statistically independent to each other and non-gaussian through a linear transformation, although unlike PCA, the components are not ordered in any way [25]. Intuitively, one can imagine that a set of signals are a combination of source signals that have been combined in some way. Equation (18) demonstrates this idea, where x_l is the l_{th} mixed signal, $s_1, \dots s_m$ are the original source signals, and $a_1, \dots a_m$ are the coefficients that transform the set of signals into the mixed signal x_l. In this application with hyperspectral images, x_l is one of the L total channels of the image and $s_{1,} \dots s_m$ are the source channels to be recovered. Although in this case there are not necessarily a true set of source channels that created the channels in the hyperspectral image, this assumption can be made to accomplish dimensionality reduction. The problem to solve then becomes estimating the a coefficients, which can be combined for all signals to form the mixing matrix, as shown in (19). The unmixing matrix that converts the mixed signals into source signals is then defined as W in equation (20) [25].

$$x_l = a_{l1}s_1 + a_{l2}s_{2+\dots} a_{lm}s_m \tag{18}$$

$$x = As \tag{19}$$

$$s = Wx \tag{20}$$

Like PCA, after the unmixing matrix W is learned on the training data, it can be applied directly to the validation and test data for preprocessing. As a preprocessing step for ICA, the

input data points are also centered and whitened. An implementation of ICA called the FastICA algorithm [25] is used here to learn the unmixing matrix W and is available in Scikit-learn's decomposition package [26].

6 EXPERIMENTAL ANALYSIS

All of the pretrained networks utilized here are publicly available in the TensorFlow standard library as a part of the Keras package [3]. The training setup allows for a maximum of 500 epochs and utilizes the early stopping method, although training rarely goes beyond 100 epochs. The early stopping method is configured such that if no improvement is made after 20 epochs, as defined by the loss on the validation set, then training will stop. The fully connected layers for each model are trained using the adam optimizer and utilize the ReLU activation function. In addition to this, each model is trained on 25 separate splits of the data to ensure that results are not greatly affected by the choice of samples in the training and test sets. The same splits of data are used for every model so that results can be compared accurately. A standard naming convention that includes the preprocessing technique and the number of channels is also used to distinguish each preprocessing method. For example, 'pca_1' is used to describe the principal component analysis method with one channel. When the number of channels used is greater than 1, 'ensemble' is appended to the description, as in 'mmca_3_ensemble', to indicate that multiple fully connected networks are used in the model. Finally, the naming convention for the ensemble of all 11 pretrained models is 'all_ensemble'.

6.1 Comparing Pretrained Models and Preprocessing Methods

Table 6 shows the average test accuracy for each pretrained model for the Pavia University and Salinas datasets. This metric is computed by averaging the test performance for each pretrained model across all preprocessing methods and sets of classes. The ResNet models are the clear winner for both datasets, with around 4% better accuracy on average than the next best models. Among the ResNet models, there is no significant difference in test accuracy which means that the depth of the network is not an important factor for this application. It is worthwhile to note that the performance difference between the best and worst models is quite

large at 12% for Pavia University and 11.8% for Salinas, which means that the choice of the model here can significantly affect the performance.

Table 6

Pretrained Models Ranked by Average Accuracy Across Class Subsets

Pavia University		Salinas	
Model	Test Accuracy	Model	Test Accuracy
ResNet50	0.738	ResNet50	0.837
ResNet101	0.733	ResNet101	0.833
ResNet152	0.726	ResNet152	0.832
DenseNet201	0.696	VGG19	0.788
DenseNet169	0.684	VGG16	0.778
DenseNet121	0.678	DenseNet121	0.757
VGG19	0.677	ResNet50V2	0.754
VGG16	0.671	DenseNet169	0.752
ResNet152v2	0.642	ResNet152V2	0.750
ResNet50v2	0.638	DenseNet201	0.749
ResNet101v2	0.618	ResNet101V2	0.719

Using the same technique as above, the preprocessing methods are ranked by their average accuracy over all data subsets and pretrained models in Table 7. On the Pavia University dataset, the top three performing methods are all within 1% accuracy of each other, indicating that they are about equal in performance, although mmca_5_ensemble is the top performer. For Salinas, rgb is the top performer with a 3.6% improvement over the next best method, kmeans_5_center_ensemble. When looking at the top five methods, Pavia University shows only a 3.3% difference in accuracy among the methods, while Salinas shows a 6.3% difference. For both datasets, mmca_5_ensemble and rgb are in the top three performing methods suggesting that they may perform well in differing datasets. The two worst performing methods are also consistent in both datasets, with kmeans_3_avg_ensemble and greyscale having the worst performance. The poor performance for kmeans_3_avg_ensemble demonstrates that creating any

ensemble of channels is not enough to achieve high accuracy, since this method can be viewed as an ensemble version of the greyscale method.

There is some variation for the rest of the methods in the middle, however. For example, mmca_3_ensemble only shows a 3.3% difference in accuracy from the best method in Pavia University, while with the Salinas dataset that increases to 11.1%. Additionally, although kmeans_5_center_ensemble performs well on average for the Salinas dataset, there is a significant decrease of 4.9% when using the same method with three channels (kmeans_3_center_ensemble). The same pattern is present for the MMCA methods, with mmca_5_ensemble having a 6.6% better accuracy on average than its 3-channel counterpart, which only has a 1.56% advantage over the greyscale method. With Pavia University, however, this difference is not as prominent.

Table 7

Preprocessing Methods Ranked by Average Accuracy Across Class Subsets

Pavia University		Salinas	
Preprocessing method	Test Accuracy	Preprocessing method	Test Accuracy
mmca_5_ensemble	0.728	rgb	0.851
rgb	0.722	kmeans_5_center_ensemble	0.815
pca_3_ensemble	0.722	mmca_5_ensemble	0.807
kmeans_5_center_ensemble	0.717	ica_3_ensemble	0.792
mmca_3_ensemble	0.695	pca_3_ensemble	0.788
kmeans_3_center_ensemble	0.689	ica_5_ensemble	0.781
ica_3_ensemble	0.680	kmeans_3_center_ensemble	0.766
pca_1	0.673	pca_1	0.747
ica_5_ensemble	0.639	mmca_3_ensemble	0.740
kmeans_3_avg_ensemble	0.631	kmeans_3_avg_ensemble	0.738
greyscale	0.604	greyscale	0.726

Although the average performance of the models and preprocessing methods show a good overview of performance, the performance of models on the full set of classes is also important. Table 8 lists the top five and bottom five models for the Pavia University dataset with all nine

classes, where the combination of pretrained model and preprocessing method are unique. As demonstrated in Table 6 and Table 6, ResNet is the best model architecture and mmca_5_ensemble is the top preprocessing method. The top five models are all within 1.8% accuracy of each other, meaning that they all share similar performance. The standard deviation of the training accuracy over 25 independent training runs is also listed and shows that there is at most a difference of 0.015 within the top five. Although the top five share similar performance, the drop in performance to the bottom five is very significant, showing an approximately 26% average difference. As expected, the ResNetV2 models and the greyscale method performed poorly and fill four out of the bottom five. This large difference in accuracy demonstrates that the choice of pretrained model and preprocessing method can have a significant impact on performance.

Table 8

Top and Bottom 5 Model Combinations on the Pavia University 9-Class Dataset

Pretrained model	Preprocessing method	Test accuracy	Test accuracy std.
Top 5			
ResNet50	mmca_5_ensemble	0.736	0.026
ResNet101	mmca_5_ensemble	0.734	0.022
ResNet50	pca_3_ensemble	0.724	0.017
ResNet50	rgb	0.718	0.032
ResNet152	mmca_5_ensemble	0.718	0.028
Bottom 5			
VGG16	greyscale	0.487	0.023
ResNet101v2	ica_5_ensemble	0.485	0.047
ResNet152v2	greyscale	0.458	0.034
ResNet50v2	greyscale	0.451	0.037
ResNet101v2	greyscale	0.417	0.028

Table 9 shows the top and bottom five combinations of pretrained model and preprocessing method on the full 16-class Salinas dataset. Again, the top five results are consistent with the results in Table 6 and Table 7. The rgb method shows a significant advantage over the

kmeans_5_center_ensemble here with 7.4% additional accuracy. This is an interesting result, because the rgb model did not show a large advantage with the Pavia University dataset. There is also a significant difference in performance of over 30% between the top and bottom 5 combinations, which is similar to that shown in the Pavia University dataset. In addition to the average test accuracy, the standard deviation of these results also shows some difference in model performance in the bottom five models. There is high standard deviation seen in the MMCA results in the bottom five, although the models there are of the worse performing DenseNet and ResNetV2 families. For Pavia University, the highest standard deviation displayed is only 0.047, while here that increases to 0.091. This high level of standard deviation compared to Pavia University could be due to the difference in number of classes, where Salinas has almost twice as many as Pavia University.

Table 9

Top and Bottom 5 Model Combinations on the Salinas 16-Class dataset

Pretrained model	Preprocessing method	Test accuracy	Test accuracy std.
Top 5			
ResNet152	rgb	0.828	0.015
ResNet50	rgb	0.817	0.018
ResNet101	rgb	0.803	0.015
ResNet101	kmeans_5_center_ensemble	0.754	0.028
ResNet50	kmeans_5_center_ensemble	0.752	0.030
Bottom 5			
ResNet152v2	greyscale	0.531	0.018
DenseNet121	mmca_3_ensemble	0.526	0.088
ResNet101v2	greyscale	0.526	0.014
DenseNet201	mmca_3_ensemble	0.526	0.091
ResNet101v2	mmca_3_ensemble	0.510	0.062

Fig. 9 shows the best models in each group (VGG, ResNet, ResNetV2, and DenseNet) on the Pavia University dataset with a varied number of classes. The best performing model with the

greyscale and rgb preprocessing methods are also included for reference. At two classes, the difference between test accuracy is no greater than 5% between all the models. As the number of classes increases, the difference quickly grows to over 10% between the best and worst models. However, for all models there is a significant decrease of over 20% when moving from 2 to 3 classes, indicating that the two-class classification problem is significantly easier for the model to understand. In spite of this, the maximum decrease in performance between the 3-9 class datasets does not go beyond 10% for the top two models ResNet50 with rgb and ResNet50 with mmca_5_ensemble model. In other models like ResNet101 with greyscale and ResNet152V2 with mmca_5_ensemble, this decrease in performance goes beyond 20%. This demonstrates that the choice of pretrained model and preprocessing method together have much greater significance than either on their own.

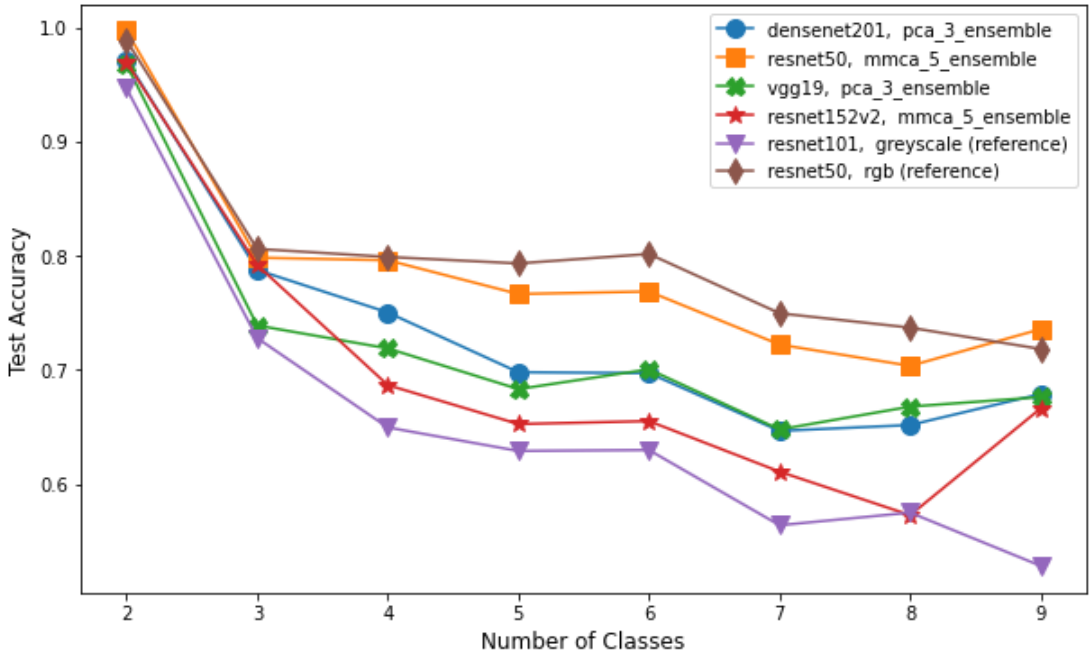

Fig. 9. The best models for each network group on the Pavia University datasets.

Fig. 10 shows the result for each subset of the Salinas data. As with Fig. 9, the best performing model of the different pretrained model groups are displayed. The pattern here is similar to the one in the Pavia University dataset, with performance differences that are small at low numbers of classes and large at higher numbers of classes. After the dataset includes six total classes, the ResNet152 model with rgb shows a clear advantage over the other models and maintains an almost 5% advantage over the next best model, ResNet50 with kmeans_5_center_ensemble. Interestingly, the reference ResNet50 greyscale model's performance is within 5% of the DenseNet121 model's performance across all the data subsets, which indicates that the choice of model here may be more important than the preprocessing method. Although the performance is mostly steadily decreasing as the number of classes increases, there is a large drop in performance when going from 13-14 classes for all models. While this seems unexpected, on further inspection this result can be explained

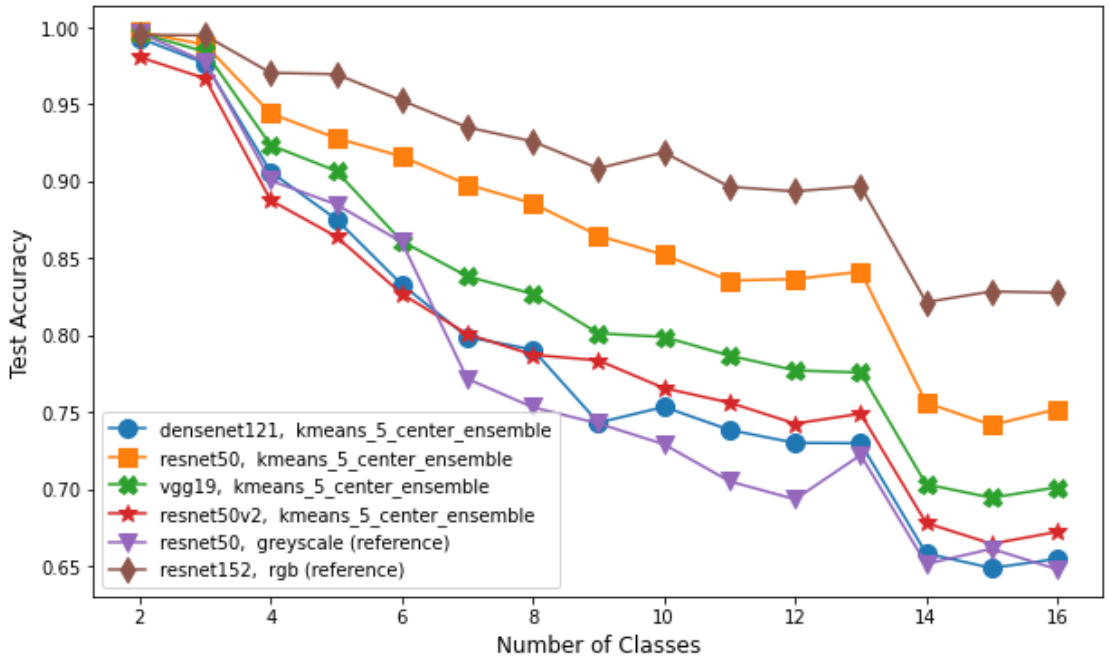

Fig. 10. The best models for each network group on the Salinas datasets.

Fig. 11 shows a sample confusion matrix for the Salinas dataset with 16 classes, where the diagonal values show the percentage of samples predicted correctly for each class (per-class accuracy), and only values greater than 0.1 are displayed. The confusion matrix shows that class 8 and 15 are often confused for each other, with class 15 being predicted for class 8 42% of the time, and class 8 being predicted for class 15 34% of the time. This pattern explains the drop in accuracy for the Salinas datasets when going from 13-14 classes, since that is the point at which both class 8 and class 15 are present in the data as shown in Table 4. Additionally, this result is found in results from other research papers. For example, [17] and [14] both list the per-class accuracy for several models on the Salinas dataset and show that class 8 and class 15 almost always have the worst performance for each model. Although the confusion matrices aren't given, it is likely that the same issue is occurring here for these classes. This also demonstrates some important points about the experimental setup, which are that the combination of classes can have a significant impact on performance, and that performance may not drop steadily as additional classes are added to the dataset. Intuitively, this makes sense, because not all classes will have the same distinguishability. For example, it will be easier to classify a dog or a cat than two different breeds of dogs, since dogs and cats have much different characteristics. In the case of the Salinas dataset, these classes are 'Grapes untrained' (8) and 'Vineyard untrained' (16), which are at the very least similar by description.

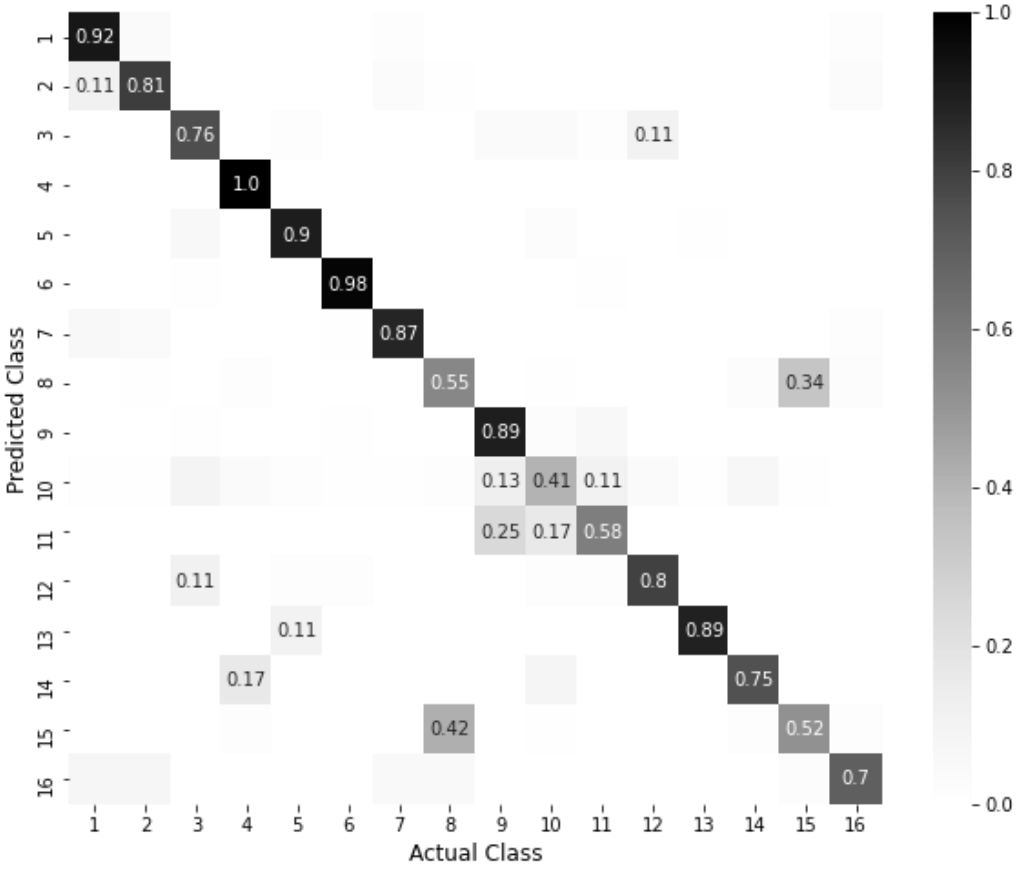

Fig. 11. Sample confusion matrix using the 16-Class Salinas dataset.

6.2 Model Ensemble Results

To further improve on the performance of the previous models, an ensemble of all 11 pretrained networks was tested for each preprocessing method. The results for each preprocessing method averaged across all sets of classes are shown for both datasets in Table 9. As is typical of ensemble methods, the ensemble shows an improvement over their single model counterparts on both datasets. Compared to the best single network models, the ensemble produces a 9.7% improvement in accuracy for Pavia University and a 7.4% improvement in accuracy for Salinas on the averaged data subsets. As with the single model networks, the difference in accuracy between the best and worst methods is significant, at 12.8% for Pavia University and 10.9% for

Salinas. For Pavia University, the top three methods are the same as in the single model results, but the order is different, with rgb as the best method instead of mmca_5_ensemble. For Salinas, however, only rgb is in the top three for both the single models and the ensemble model, although the difference in the top five methods is only 3.5%.

Table 10

Preprocessing Methods Ranked by Average Accuracy Across Class Subsets (all_ensemble)

Pavia University		Salinas	
Preprocessing method	Accuracy	Preprocessing Method	Accuracy
rgb	0.833	rgb	0.925
pca_3_ensemble	0.832	ica_5_ensemble	0.915
mmca_5_ensemble	0.819	ica_3_ensemble	0.912
kmeans_5_center_ensemble	0.808	pca_3_ensemble	0.905
ica_3_ensemble	0.807	kmeans_5_center_ensemble	0.890
mmca_3_ensemble	0.799	mmca_5_ensemble	0.889
ica_5_ensemble	0.790	kmeans_3_center_ensemble	0.874
kmeans_3_center_ensemble	0.788	mmca_3_ensemble	0.867
pca_1	0.755	pca_1	0.830
kmeans_3_avg_ensemble	0.727	kmeans_3_avg_ensemble	0.829
greyscale	0.705	greyscale	0.816

The results for the best preprocessing methods in each category (greyscale, RGB, ICA, K-means, MMCA, and PCA) are shown in Fig. 12 for the Pavia University datasets. With more than three classes, the difference between the greyscale method and the other methods is 9% at minimum. However, the difference between the rest of the preprocessing methods is less significant and does not go above 5% for most of the data subsets. This finding is consistent with the analysis from the single network models and preprocessing methods, which is that the choice of preprocessing method may be less significant than the choice of model.

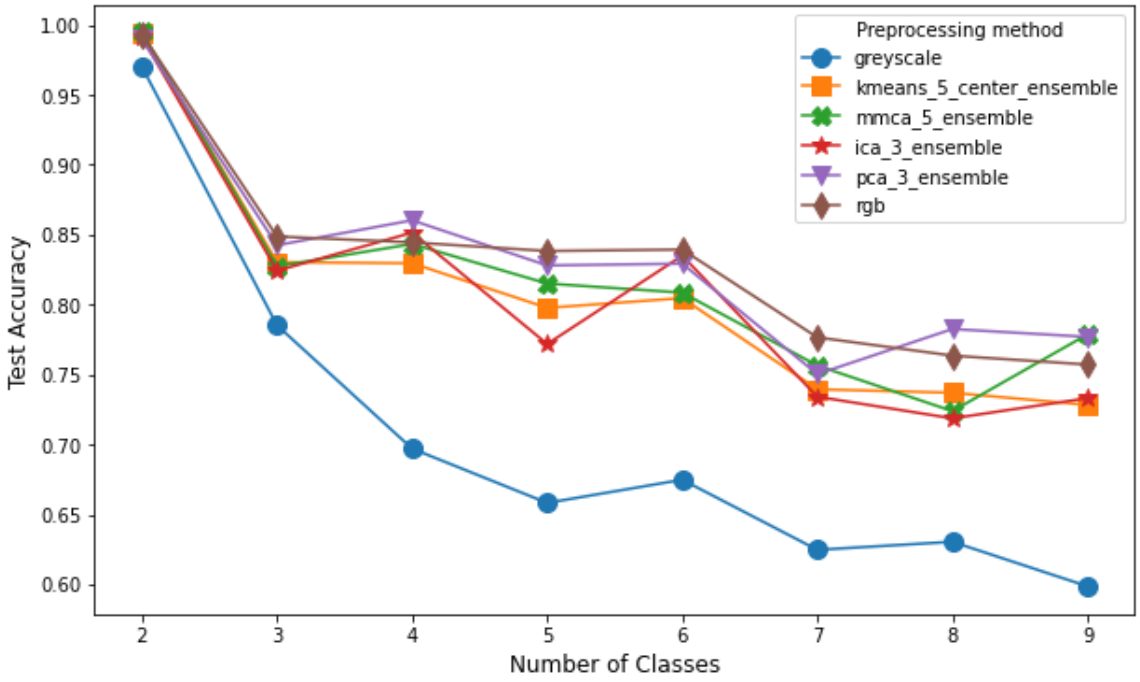

Fig. 12. The best preprocessing methods in each group with the all_ensemble model (Pavia University).

Fig. 13 shows the results for the all_ensemble models with different numbers of classes for the Salinas dataset. The best preprocessing methods from each category are displayed along with the greyscale and rgb methods. For datasets with 2-3 classes, the choice of preprocessing method is insignificant, as even the greyscale method is on par with the other methods. Once the number of classes reaches 4 and above, the performance of the greyscale method declines rapidly, while all other techniques maintain an accuracy between 85-95% until the drop at 13-14 classes that was described previously in Section 6.1. For all methods except greyscale, the difference in performance in the range of 4 to 13 classes is no more than 10%, while for the best single model network in this range there is up to an 11% difference (see Fig. 10). Additionally, both the single and ensemble models start at around 94% accuracy for three classes, signifying that the ensemble method is able to maintain a high accuracy as the complexity of the dataset increases.

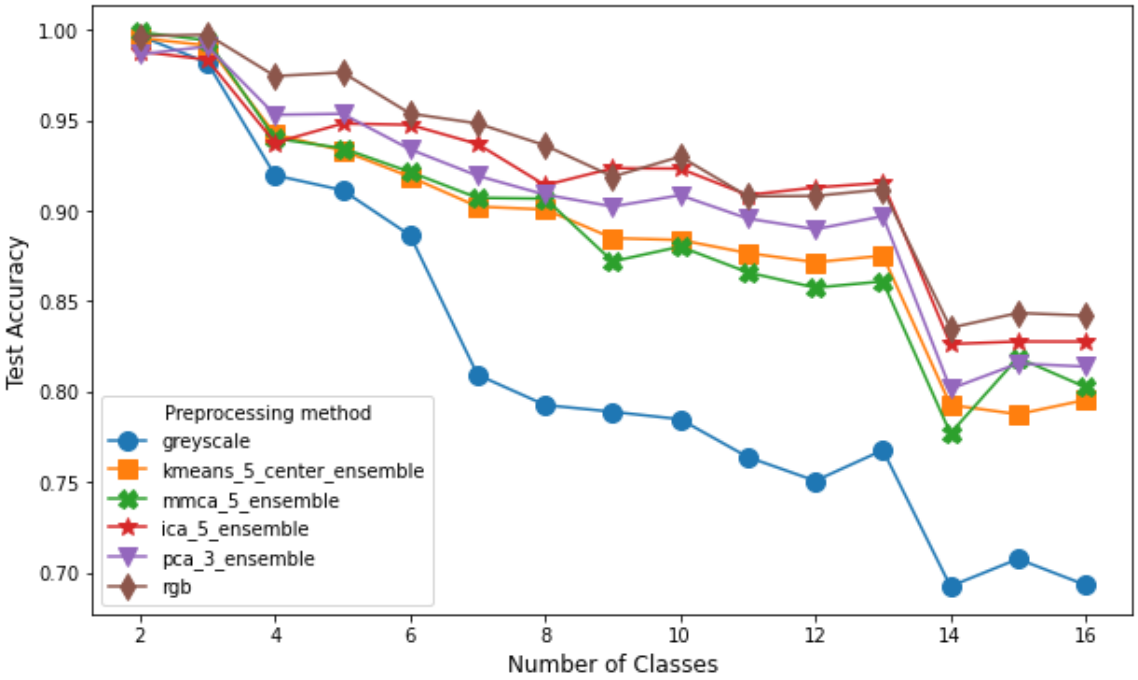

Fig. 13. The best preprocessing methods in each group with the all_ensemble model (Salinas).

7 FUTURE WORK

The methods implemented here provide opportunity for future work. One of the main areas for exploration is the area of ensemble models. While the work here delved into the area of ensemble models by combining multiple pretrained networks and by using multiple output networks for different channels, there is still room for optimization. Combinations of models and channels could be intelligently chosen through validation set performance which could help eliminate any poorly performing models that may hinder the performance of the ensemble. Additionally, multiple preprocessing methods could be combined to create an ensemble of models with a single pretrained network. For example, PCA and MMCA could be applied to create two separate datasets and each of these datasets could be used to train an output network for a given pretrained model. Additionally, many more channels could be used as part of the ensemble. The techniques described in [16] show how including a large number of channels can improve the performance of different models, although there are diminishing returns as the number of channels increases. Since the 5-channel ensembles worked well here, it is likely that including additional bands would improve performance of the model.

Other methods of feature selection and feature extraction can also be explored. These could include deep-learning based approaches like auto-encoders that extract features from the hyperspectral data. Auto-encoders attempt to create a lower-dimensional representation of the input data using a neural network and then attempt to transform that data back into its original form. Since the coding and decoding of the data are both optimized by training, the encoded version of the data may be able to accurately represent the data in a lower-dimensional form. This lower-dimensional data could then be used in the same manner as the other preprocessing methods implemented in this paper. Using this type of method could potentially provide an advantage over channel selection methods since it is attempting to optimize the lower-

dimensional set of data in a way that allows most of its original information to be recovered, which could result in useful data for training.

Finally, optimization of the network architecture could be explored. Because results show that the ResNet model outperformed the rest of the models, it is plausible that the network structure plays a role in the performance. The authors of [14] and [17] were able to alter the pretrained networks in ways that still utilized their pretrained weights, and their results showed that their techniques were viable. Adjusting the number of layers or the shape of the layers could be simple techniques to use that may result in improved performance. The fully connected output layers are one area that would be simple to optimize. One way to do this would be to add more fully connected layers at the end of the network and adjust the number of nodes in each of these layers. Both of these parameters could be optimized on the validation set to find the optimal network structure.

Although results show that overall classification accuracy tends to decrease as the number of classes increases, a more appropriate technique for varying the numbers classes may be to take the average result of every possible class combination. For example, in a three-class situation where only two classes are being used there would be three possible combinations of two classes to use (1&2, 1&3, and 2&3). This would limit the effect that any single combination of classes has on a model's performance, since it is not true that every set of classes has equal distinguishability, as seen with classes 8 and 15 in the Salinas dataset. The drawback here, however, is that the number of possible combinations greatly increases as the number of classes increases, which significantly increases the time needed to train models. For example, choosing all combinations of five out of nine classes results in 126 unique sets of classes, which means 126 models would need to be trained. Without the proper hardware or cloud resources, the time to

train these models could become unmanageable, although it would give the most accurate estimation of actual test performance.

8 CONCLUSIONS

This work illustrated that leveraging pretrained RGB networks for hyperspectral imaging can be useful when the feature extraction methods and pretrained networks are chosen appropriately. The choice of feature extraction was shown to be an important factor in achieving high classification performance, with results differing by over 20% when comparing the best and worst methods with a given pretrained model. When possible, simply selecting the RGB bands from a hyperspectral image may offer the best performance, but when it is not possible, more complex methods may be able to match this performance. This same effect was found with the pretrained networks, which showed that the ResNet networks had the best performance, and interestingly, their ResNetV2 counterparts had the worst performance on both datasets. The best performing models were able to achieve 73.6% accuracy on the Pavia University dataset and 82.8% accuracy on the Salinas dataset with 25 samples per class when both datasets included all classes. These results show both the preprocessing method and the choice of pretrained model can greatly affect performance, and both must be optimized to achieve the best performance.

The ensemble techniques here showed that they are effective in improving performance over other non-ensemble techniques. For example, the pca_3_ensemble technique achieved 4-5% improvements in accuracy over the pca_1 technique for both datasets while comparing the average accuracy across class subsets. Increasing the number of channels even more may be a promising method to improve performance. The ensemble of all 11 models also showed additional improvements in accuracy of 4.4% for Pavia University and 7.6% for Salinas on the full class datasets. Since this included all pretrained models, its likely that optimizing the set of models can provide additional improvements.

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[2] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *International Conference on Learning Representations (ICLR)*, 2015

[3] "Module: tf.keras.applications: TensorFlow Core v2.3.0," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications. [Accessed: 01-Nov-2020].

[4] H. Zhang, Y. Li, Y. Jiang, P. Wang, Q. Shen, and C. Shen, "Hyperspectral Classification Based on Lightweight 3-D-CNN With Transfer Learning," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 8, pp. 5813–5828, 2019.

[5] "AVIRIS - Airborne Visible / Infrared Imaging Spectrometer," *NASA*. [Online]. Available: https://aviris.jpl.nasa.gov/. [Accessed: 01-Nov-2020].

[6] "Hyperspectral Remote Sensing Scenes," *Grupo de Inteligencia Computacional (GIC)*. [Online]. Available: http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes. [Accessed: 9-Oct-2020].

[7] R. Piiroinen, J. Heiskanen, E. Maeda, A. Viinikka, and P. Pellikka, "Classification of Tree Species in a Diverse African Agroforestry Landscape Using Imaging Spectroscopy and Laser Scanning," *Remote Sensing*, vol. 9, no. 9, p. 875, 2017.

[8] K. Loggenberg, A. Strever, B. Greyling, and N. Poona, "Modelling Water Stress in a Shiraz Vineyard Using Hyperspectral Imaging and Machine Learning," *Remote Sensing*, vol. 10, no. 2, p. 202, Jan. 2018.

[9] G. Lu and B. Fei, "Medical hyperspectral imaging: a review," *Journal of Biomedical Optics*, 20-Jan-2014. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/24441941/. [Accessed: 01-Nov-2020].

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] Y. Jiang, Y. Li, and H. Zhang, “Hyperspectral Image Classification Based on 3-D Separable ResNet and Transfer Learning,” *IEEE Geoscience and Remote Sensing Letters*, vol. 16, no. 12, pp. 1949–1953, Dec. 2019.

[14] L. Jiao, M. Liang, H. Chen, S. Yang, H. Liu, and X. Cao, “Deep Fully Convolutional Network-Based Spatial Distribution Prediction for Hyperspectral Image Classification,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 10, pp. 5585–5599, Oct. 2017.

[15] “Visual Object Classes Challenge 2011 (VOC2011).” [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2011/index.html. [Accessed: 01-Nov-2020].

[16] W. Sun and Q. Du, “Hyperspectral Band Selection: A Review,” *IEEE Geoscience and Remote Sensing Magazine*, vol. 7, no. 2, pp. 118–139, 2019.

[17] S. Mei, J. Ji, J. Hou, X. Li, and Q. Du, “Learning Sensor-Specific Spatial-Spectral Features of Hyperspectral Images via Convolutional Neural Networks,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 8, pp. 4520–4533, Aug. 2017.

[18] B. Kunkel, F. Blechinger, R. Lutz, R. Doerffer, H. V. D. Piepen, and M. Schroder, “ROSIS (Reflective Optics System Imaging Spectrometer) - A Candidate Instrument For Polar Platform Missions,” *Optoelectronic Technologies for Remote Sensing from Space*, vol. 0868, Apr. 1988.

[19] “scipy.ndimage.zoom: SciPy v1.5.3 Reference Guide,” *SciPy*. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.zoom.html. [Accessed: 01-Nov-2020].

[20] “sklearn.cluster.KMeans,” *Scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html. [Accessed: 01-Nov-2020].

[21] T.-M. Tu, C.-H. Chen, J.-L. Wu, and C.-I. Chang, “A fast two-stage classification method for high-dimensional remote sensing data,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 36, no. 1, pp. 182–191, Jan. 1998.

[22] C.-I. Chang, Q. Du, T.-L. Sun, and M. Althouse, “A joint band prioritization and band-decorrelation approach to band selection for hyperspectral image classification,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 37, no. 6, pp. 2631–2641, Nov. 1999.

[23] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning with applications in R*. New York: Springer, 2017.

[24] “sklearn.decomposition.IncrementalPCA,” *Scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.IncrementalPCA.html. [Accessed: 01-Nov-2020].

[25] A. Hyvärinen and E. Oja, “Independent component analysis: algorithms and applications,” *Neural Networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

[26] “sklearn.decomposition.FastICA,” *Scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html. [Accessed: 01-Nov-2020].