ENGO 697

Remote Sensing Systems and Advanced Analytics

Session 12: Hyperspectral image classification and spectral unmixing

Dr. Linlin (Lincoln) Xu
Linlin.xu@ucalgary.ca
Office: ENE 221

Outline

➔ Hyperspectral Basics

➔ Hyperspectral Image Processing Tasks

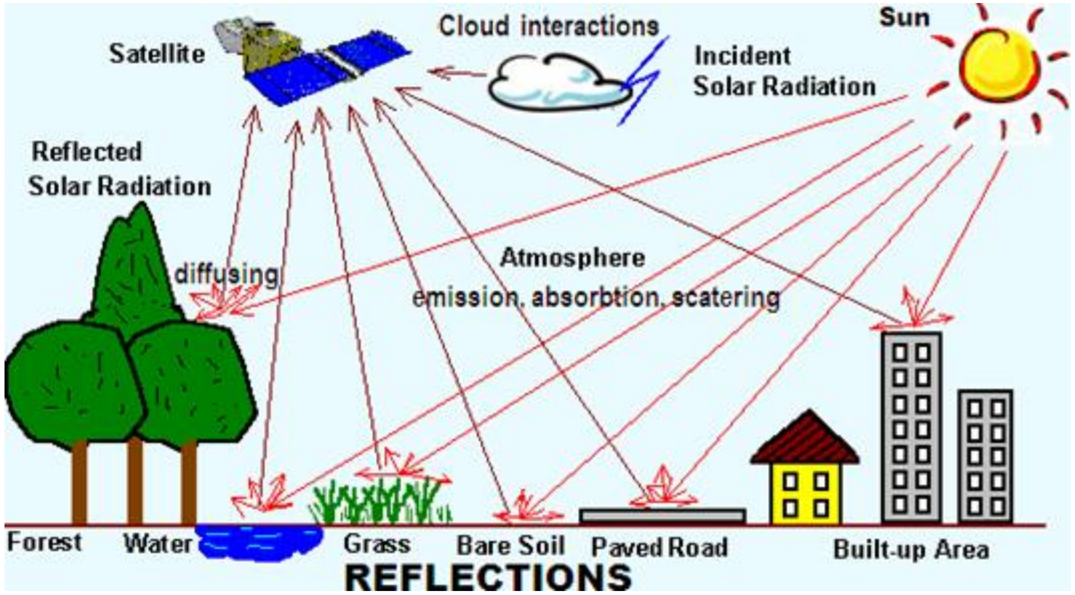
➔ UAV Hyperspectral Crop and Soil Mapping

➔ Classification vs. Spectral Unmixing

➔ Hyperspectral Unmixing

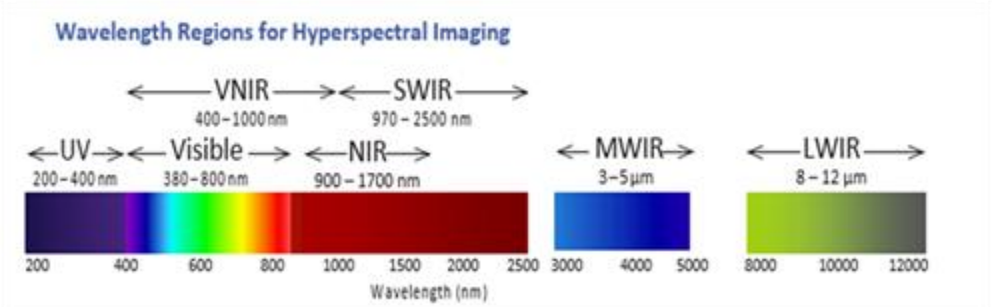
➔ Hyperspectral Image Classification

Hyperspectral Remote Sensing System

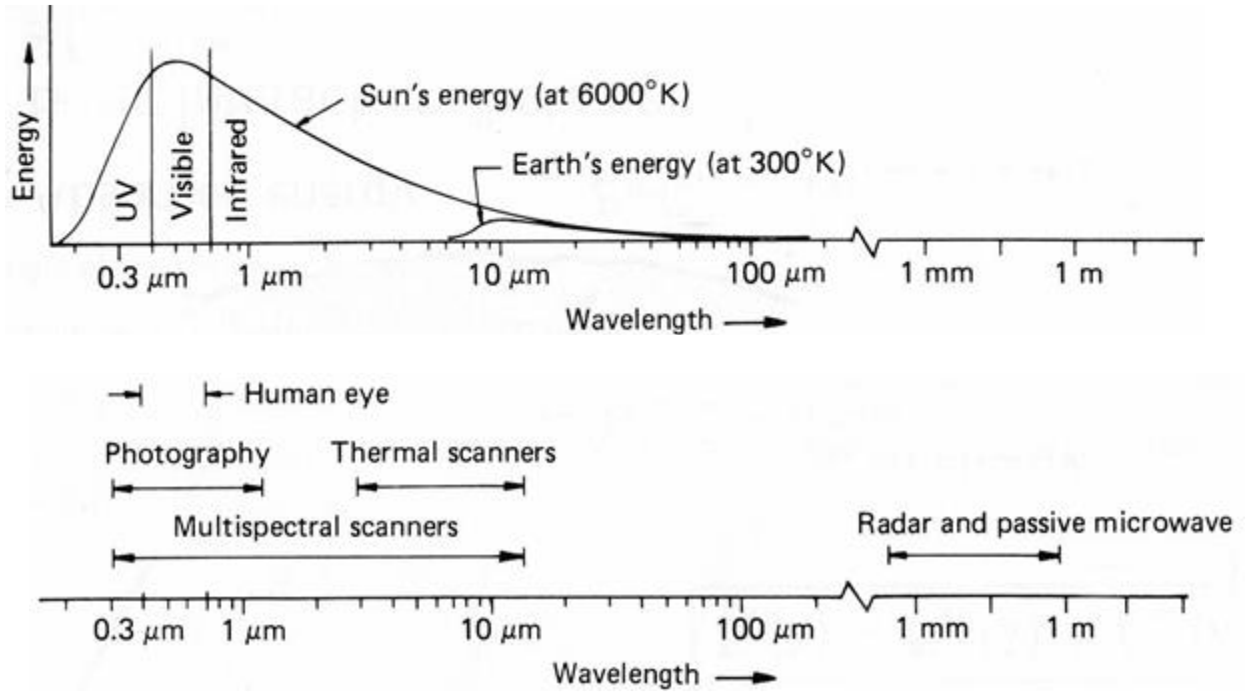


Characteristics of hyperspectral remote sensing systems:

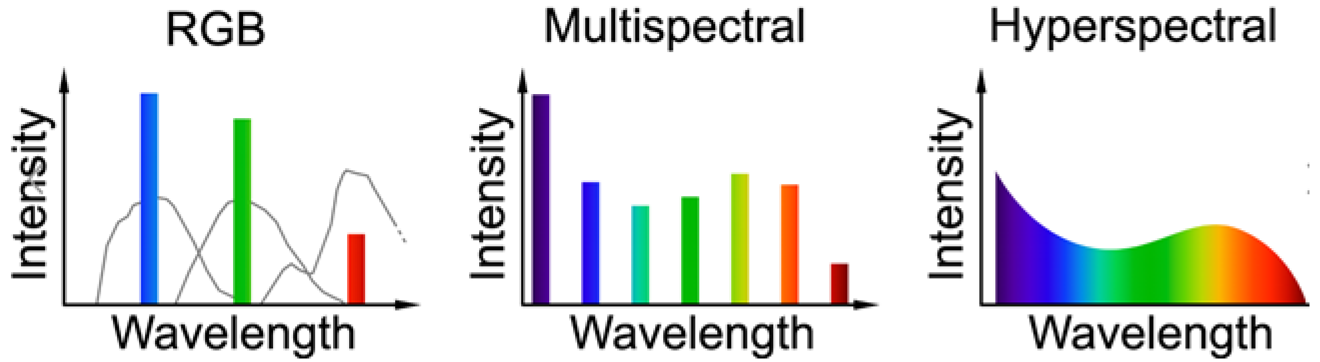
(1) Passive, relies on the Sun as the source of radiation.
(2) 400nm - 2500nm for most commercial hyperspectral sensors.
(3) Various spectral channels
(4) Trade-off between spectral resolution and spatial resolution
(5) Mixed pixels



Spectral Characteristics of Energy Sources and Sensing Systems



RGB sensors vs. Multi/Hyperspectral Sensors



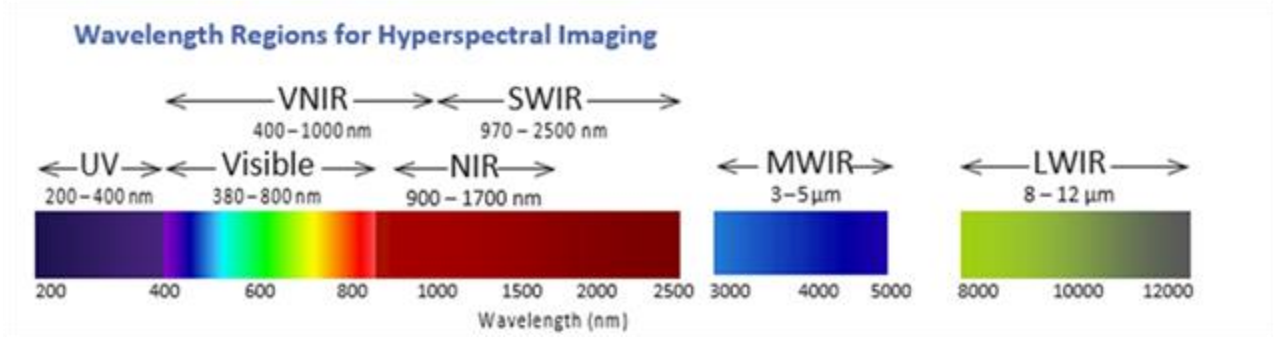


Image from wiki.tum.de and middletonspectral.com

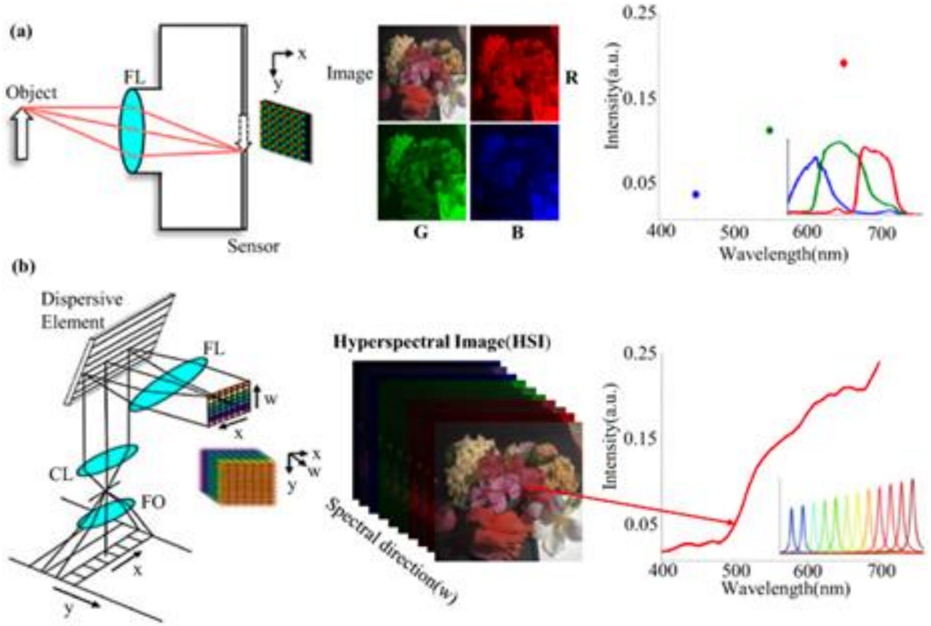
RGB sensors have only three visible channels (i.e., R, G, B).

Multispectral sensors have more than 3 channels at VNIR and SWIR portions of the spectrum (400nm-2500nm).

Hyperspectral sensors typically have hundreds of continuous channels at VNIR and SWIR portions of the spectrum (400nm-

Trade-off Between Spectral Resolution and Spatial Resolution

Is it possible for 15cm Maxar camera to have hundreds of hyperspectral channels?



Images from blog.maxar.com, nature.com

Mixed Pixel in Hyperspectral Remote Sensing Image

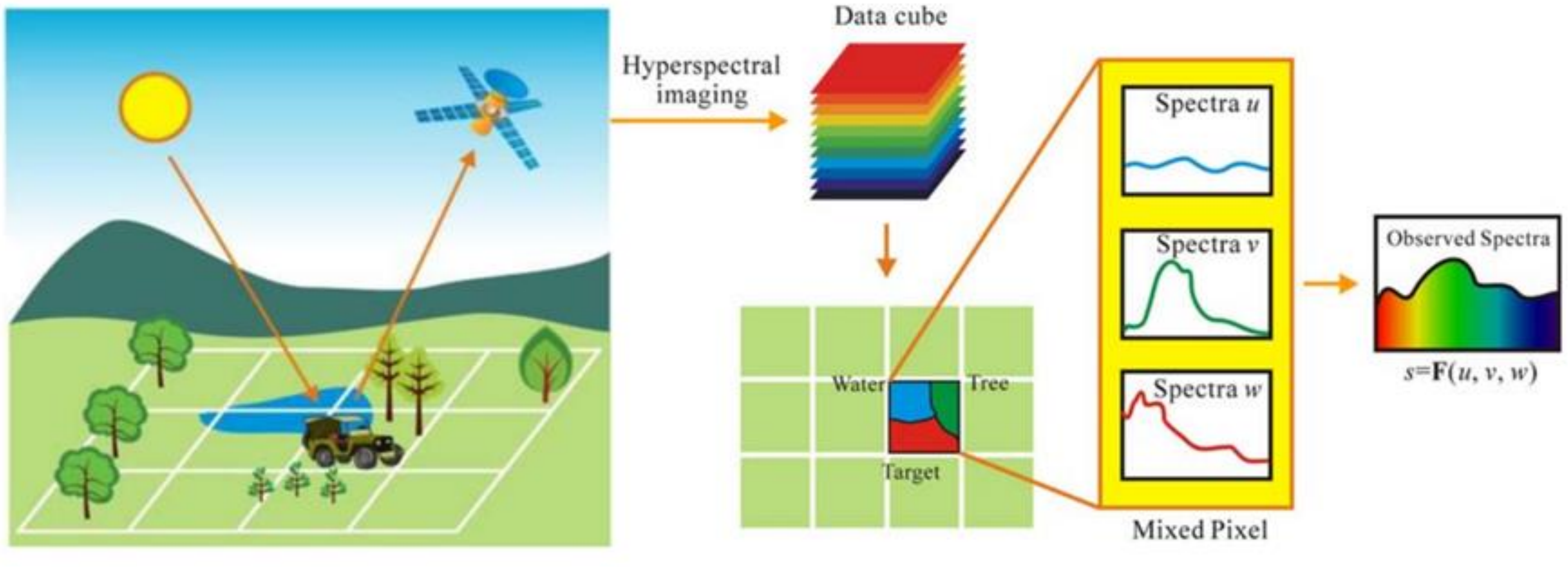
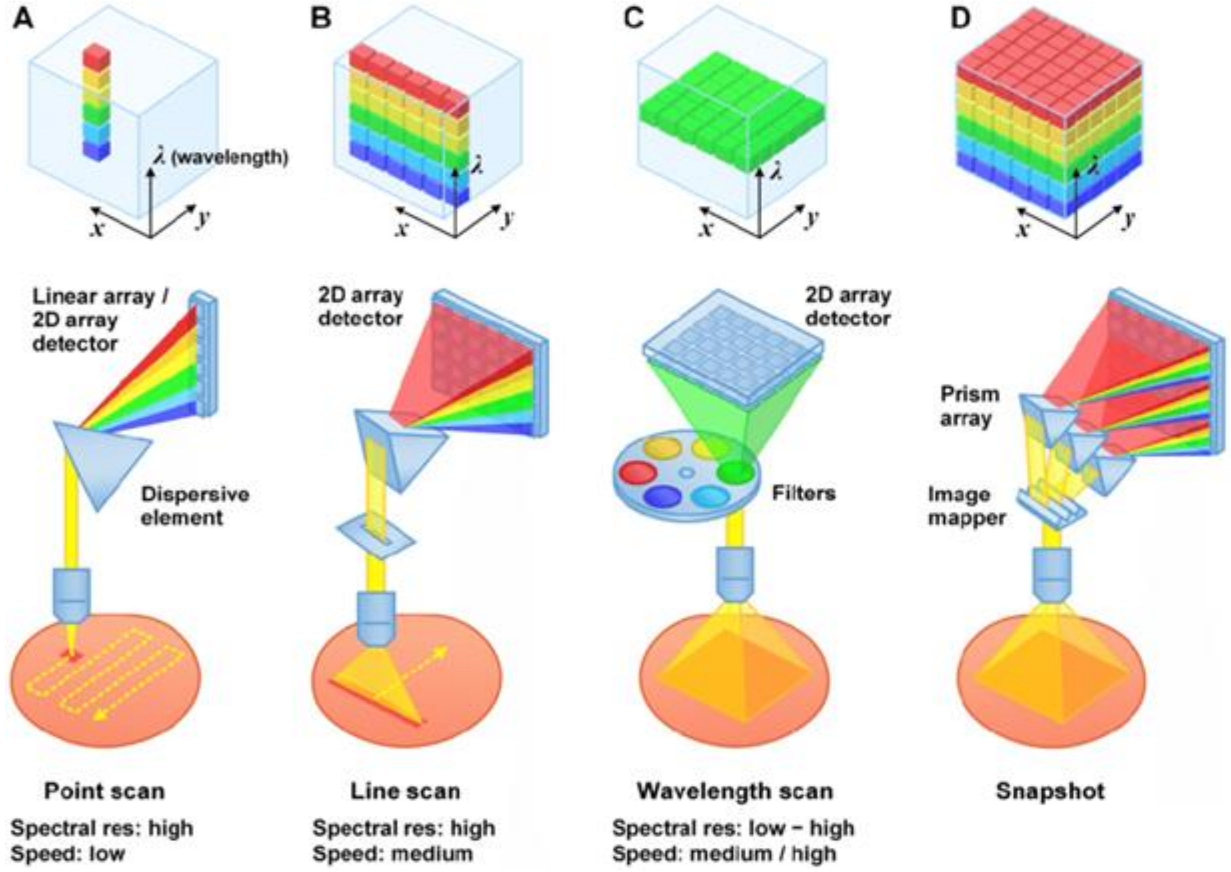
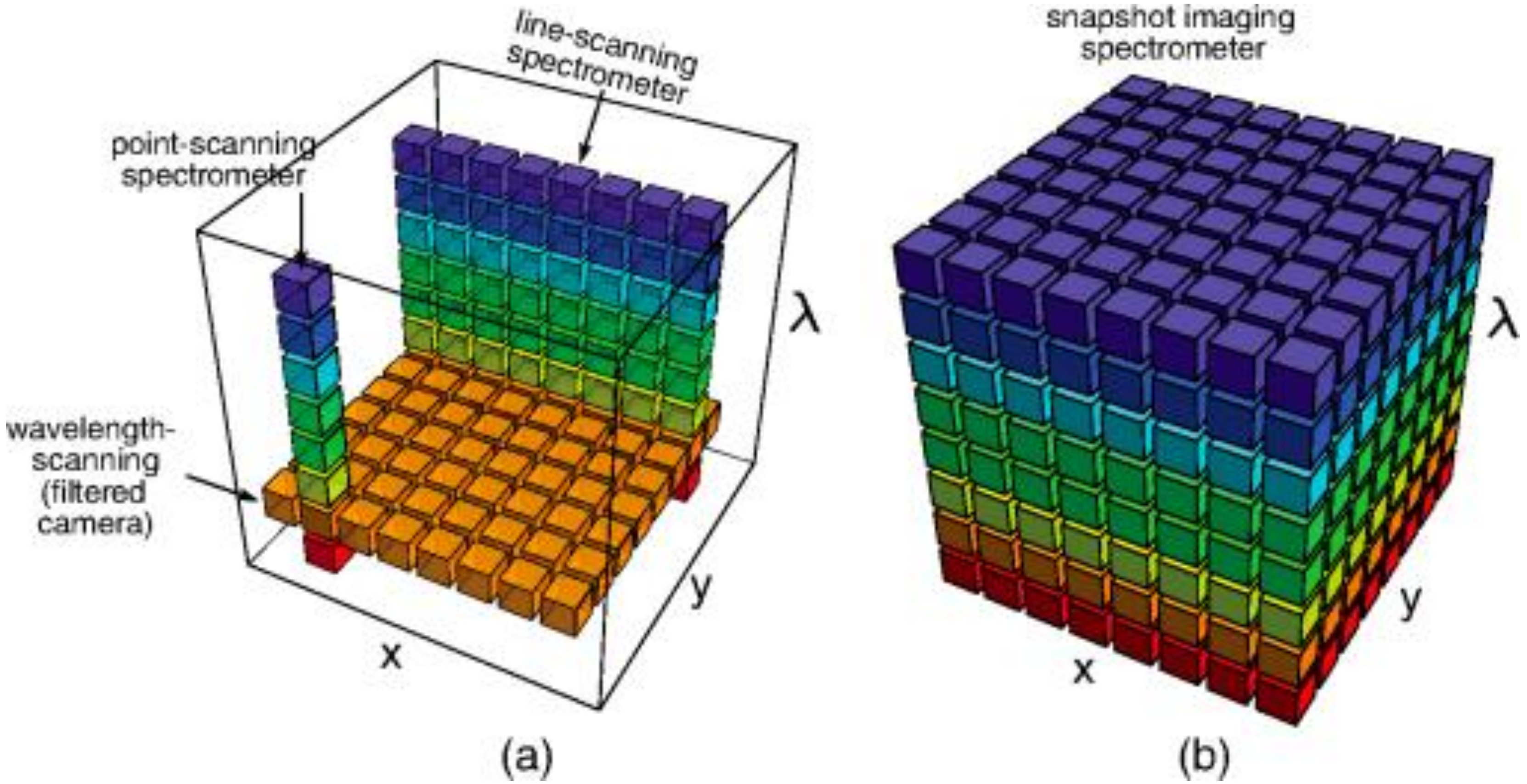


Illustration of mixed pixel generation in hyperspectral remote sensing (from Zhang et al. 2014)

Hyperspectral Imaging Approaches



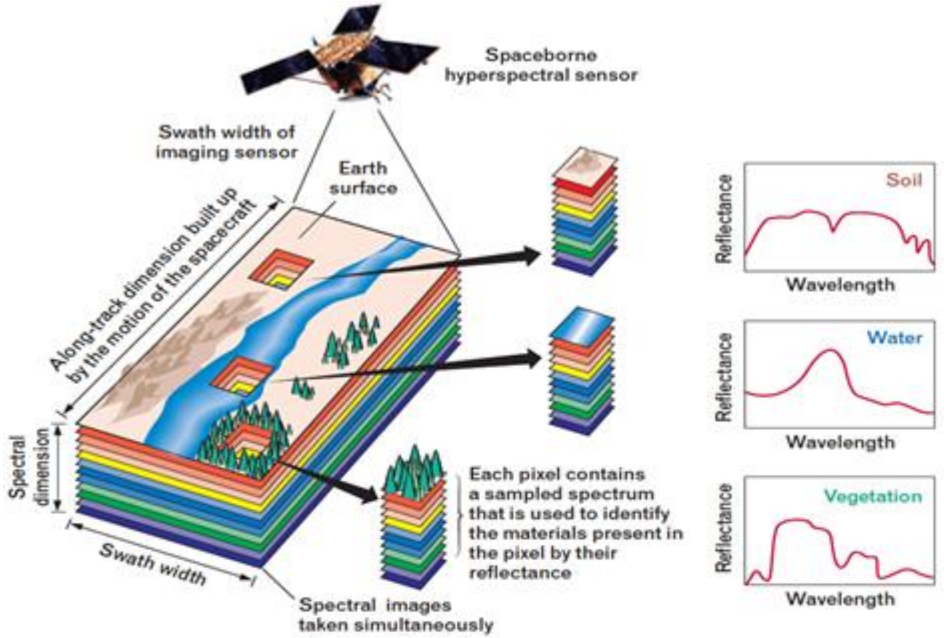
(A) Point scan. (B) Line scan (i.e. ”pushbroom”). (C) Wavelength scan. (D) Snapshot.


line-scanning spectrometer
point-scanning spectrometer
wavelength-scanning (filtered camera)
λ
y
x
(a)
snapshot imaging spectrometer
λ
y
x
(b)

https://www.spiedigitallibrary.org/journals/optical-engineering/

Outline

Hyperspectral Environmental Monitoring Analytics



Spaceborne hyperspectral sensor
Swath width of imaging sensor
Earth surface
Along-track dimension built up by the motion of the spacecraft
Spectral dimension
Swath width
Each pixel contains a sampled spectrum that is used to identify the materials present in the pixel by their reflectance
Spectral images taken simultaneously
Reflectance
Wavelength
Soil
Water
Vegetation

Image Source http://www.markelowitz.com

What do we want from hyperspectral image (HSI)?

-- ***informative features*** extraction for visualization

-- ***subtle class labels***, e.g., different crop types mapping, diseased and healthy crops discrimination;

-- ***biochemical parameters***, e.g., chlorophyll content and water content in leaves;

-- ***biophysical parameters***, e.g., leaf area index (LAI)

-- ***geochemical parameters***, e.g., soil heavy metal concentration, soil moisture;

Difficulties:

-- the large ***data volume*** of hyperspectral image (HSI);

-- the innate ***high-dimensionality*** of HSI;

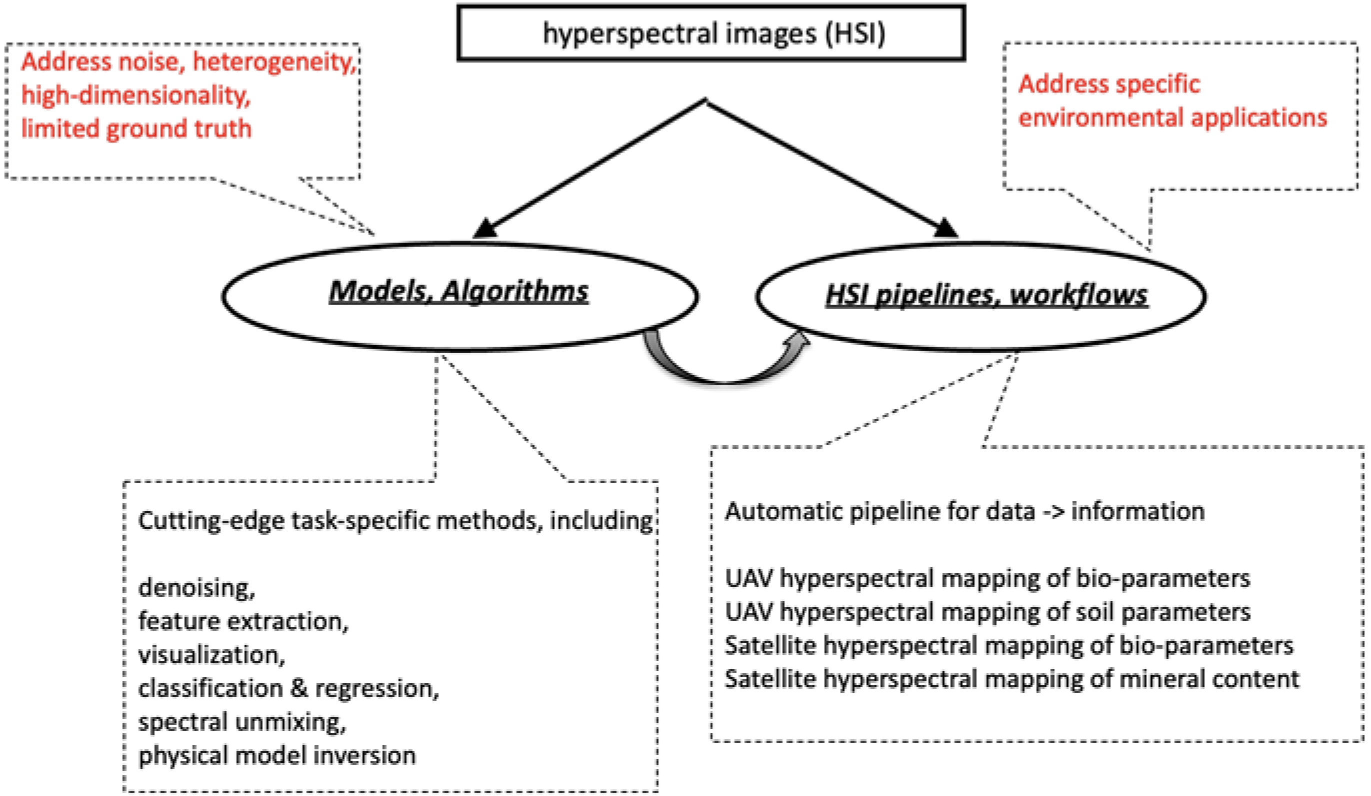
-- the ***spatial-spectral heterogeneity*** in HSI;

-- the ***limited training samples***;

-- the ***noise effect*** in HSI, and many other factors;

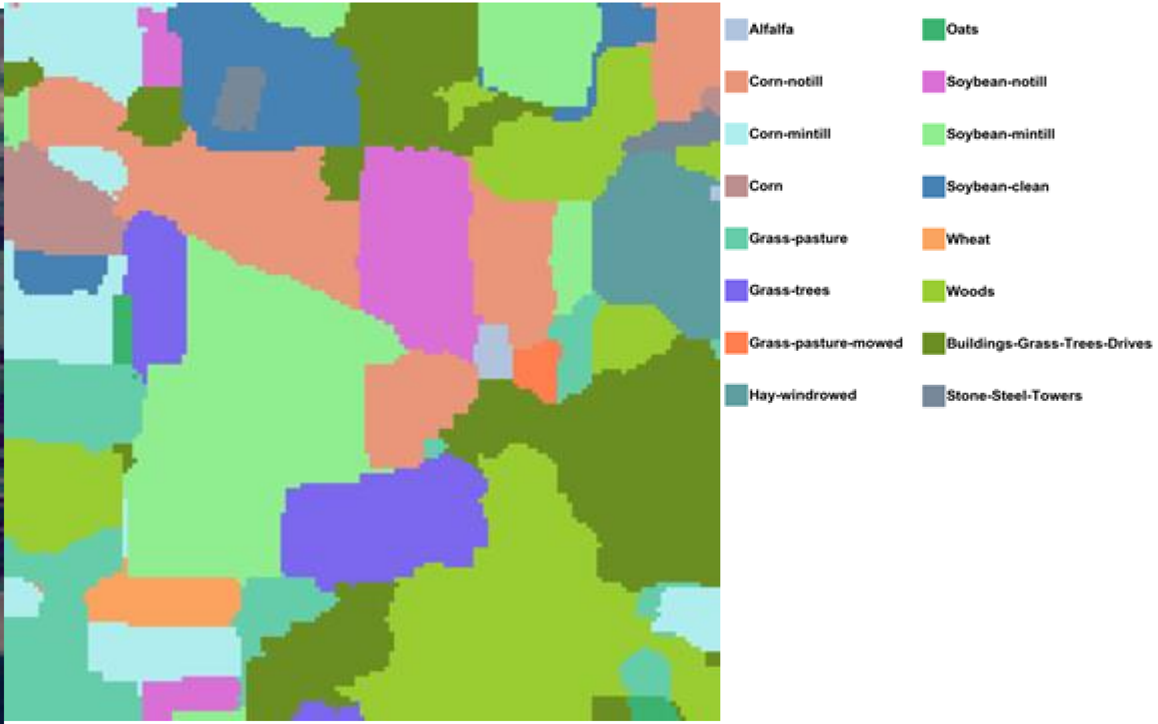
How to use ***Advanced Intelligent Machine Learning and Statistical Approaches*** to improve environmental variable extraction?

Intelligent Hyperspectral Environmental monitoring analytics

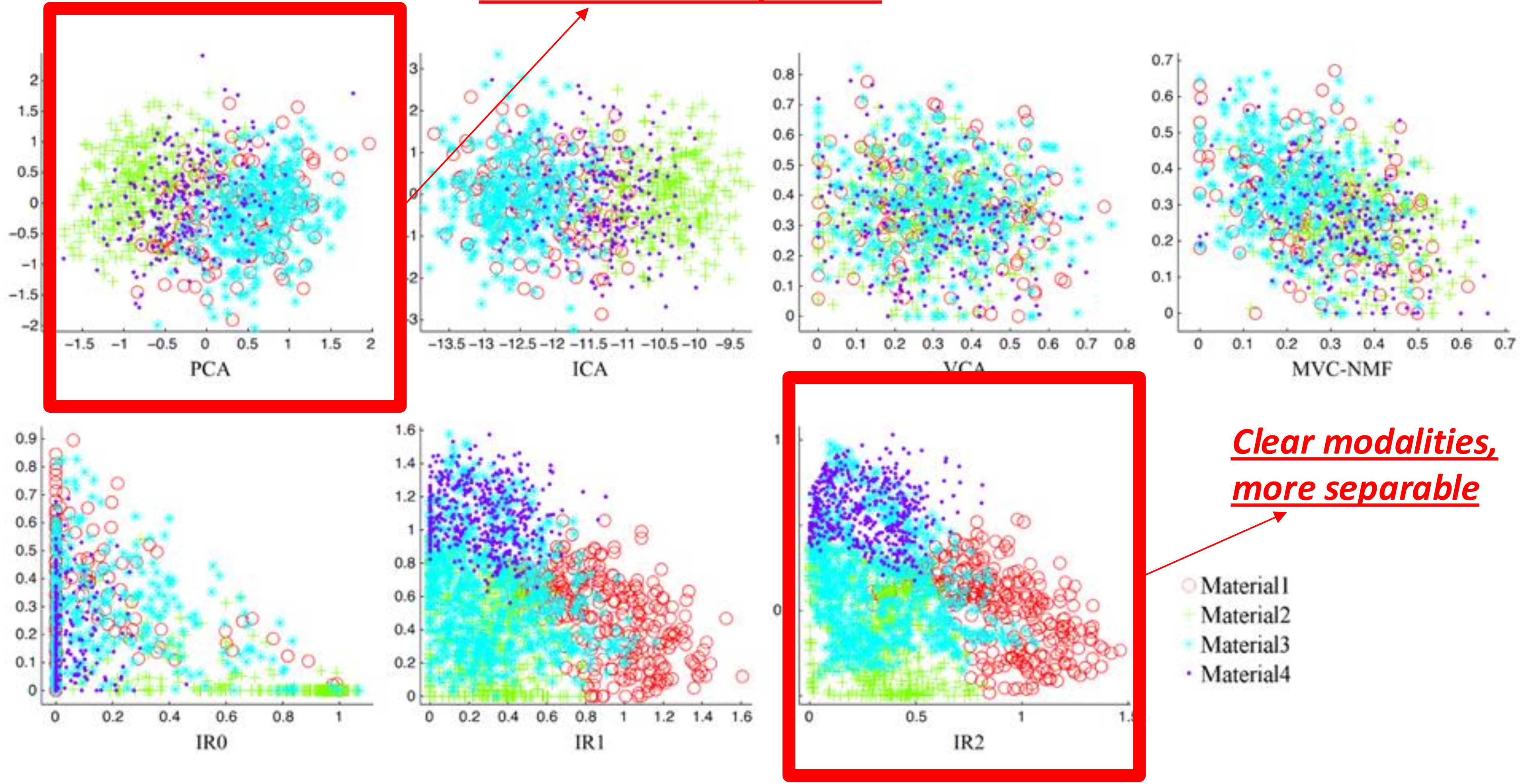

hyperspectral images (HSI)
Address noise, heterogeneity, high-dimensionality, limited ground truth
Address specific environmental applications
Models, Algorithms
HSI pipelines, workflows
Cutting-edge task-specific methods, including
denoising,
feature extraction,
visualization,
classification & regression,
spectral unmixing,
physical model inversion
Automatic pipeline for data -> information
UAV hyperspectral mapping of bio-parameters
UAV hyperspectral mapping of soil parameters
Satellite hyperspectral mapping of bio-parameters
Satellite hyperspectral mapping of mineral content

Crop classification Using Deep Learning and Spatial Modeling AVIRIS

Hyperspectral Image with 224 Channels -- North-western Indiana, two-thirds **agriculture**, and one-third **forest** or other natural perennial **vegetation**



Spatial Modeling and Deep neural network classifier achieved ***97.45%*** overall accuracy using limited training samples;

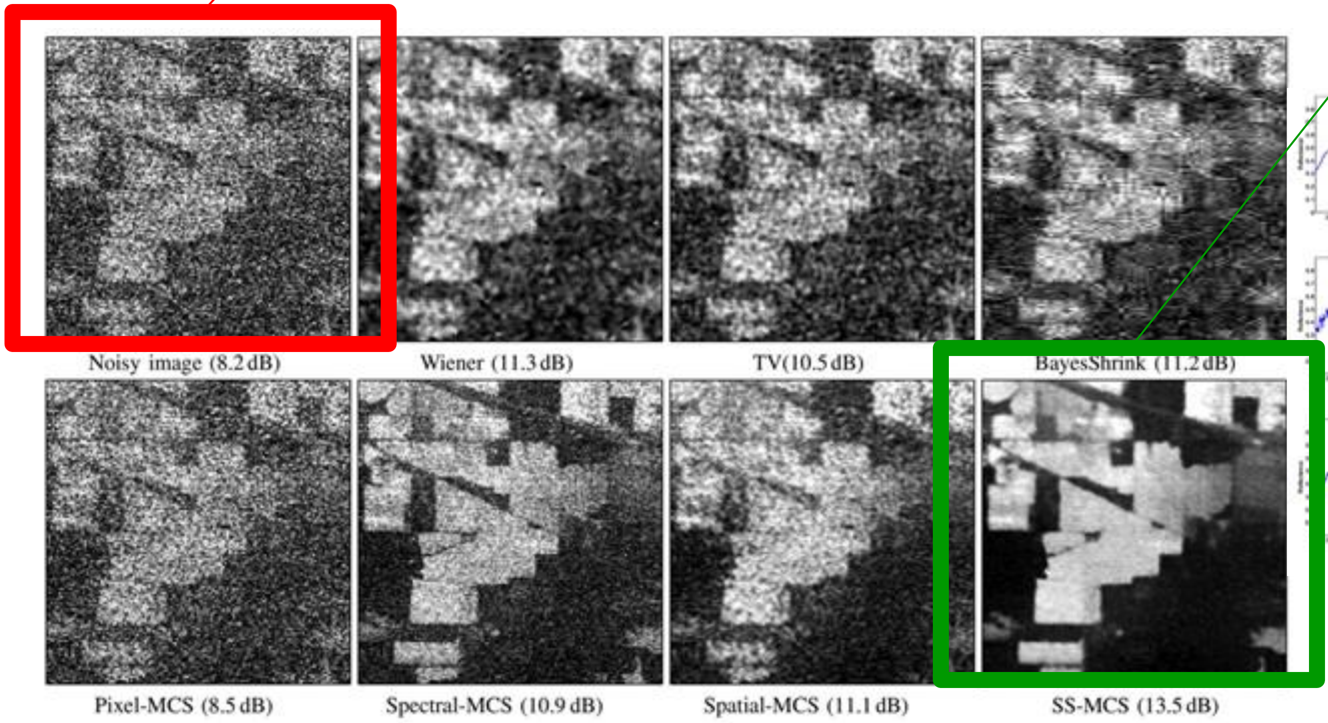

Classes are not separable
Clear modalities, more separable
PCA
ICA
VCA
MVC-NMF
IR0
IR1
IR2
Material1
Material2
Material3
Material4

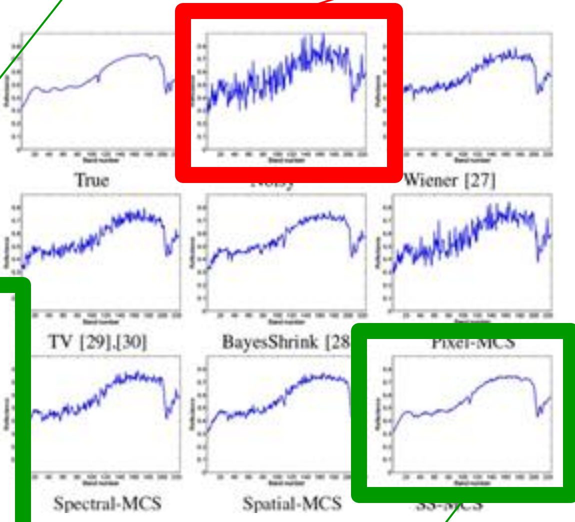
Raw noisy hyperspectral band image

Denoised hyperspectral band image

Denoising of Hyperspectral Crop Scene

Noisy spectra





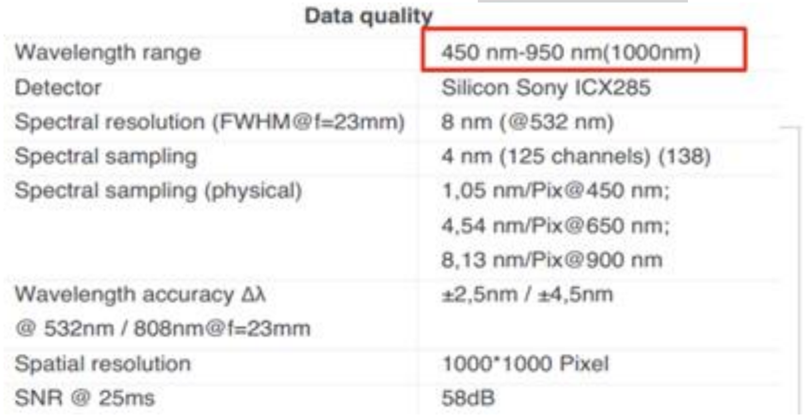
Denoised spectra

Fig. 20. Denoising results achieved by different methods, on band 219 of Indian Pines image. The SNR values are shown in parenthesis. The proposed SS-MCS method increases the SNR of noisy image dramatically by 5.3 dB. Moreover, it recovers the scene signal from intense noise pollution. Using information in adjacent channels, spectral-MCS also highlights the signals, but also preserves large amount of noise.

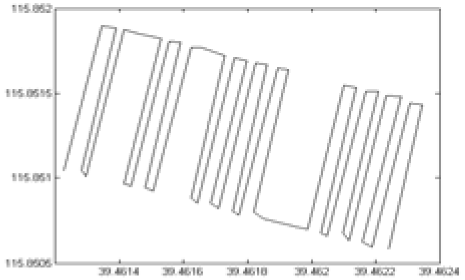
Outline

UAV Hyperspectral Canopy Monitoring

Time: June 2017
Study area： 1000m^2
Flight altitude: 50m
Flight speed: 2m/s
Forward overlap: 80%
Side overlap: 45%
Volume： 3590 images (13G)
Number of leaf samples: 30
(cab- µg/cm2, cw-g/cm2)



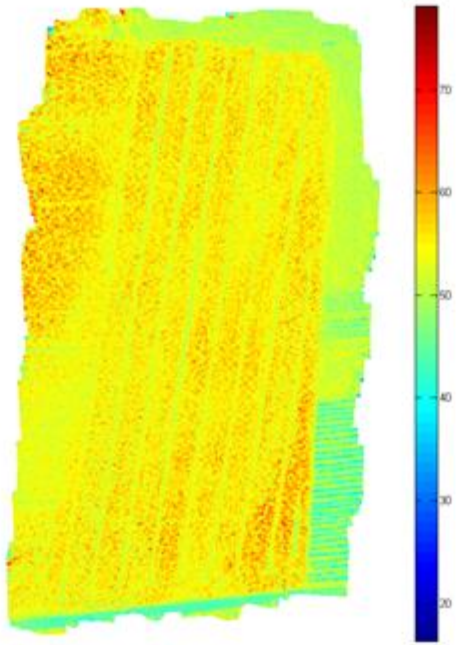
Data quality	
Wavelength range	450 nm-950 nm(1000nm)
Detector	Silicon Sony ICX285
Spectral resolution (FWHM@f=23mm)	8 nm (@532 nm)
Spectral sampling	4 nm (125 channels) (138)
Spectral sampling (physical)	1,05 nm/Pix@450 nm; 4,54 nm/Pix@650 nm; 8,13 nm/Pix@900 nm
Wavelength accuracy Δλ @ 532nm / 808nm@f=23mm	±2,5nm / ±4,5nm
Spatial resolution	1000*1000 Pixel
SNR @ 25ms	58dB



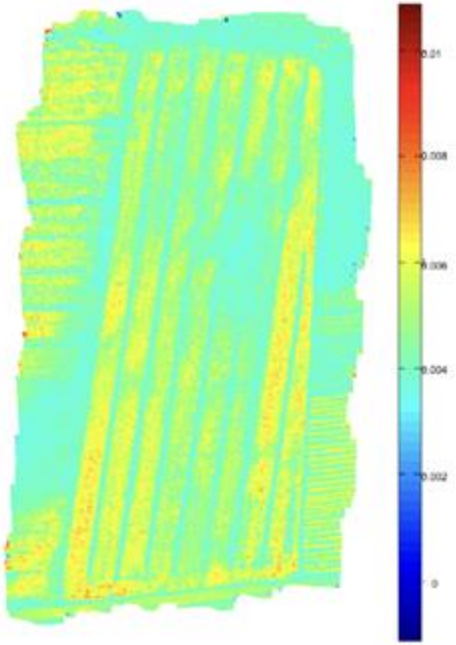
Flight track

Mosaiced image

RGB image



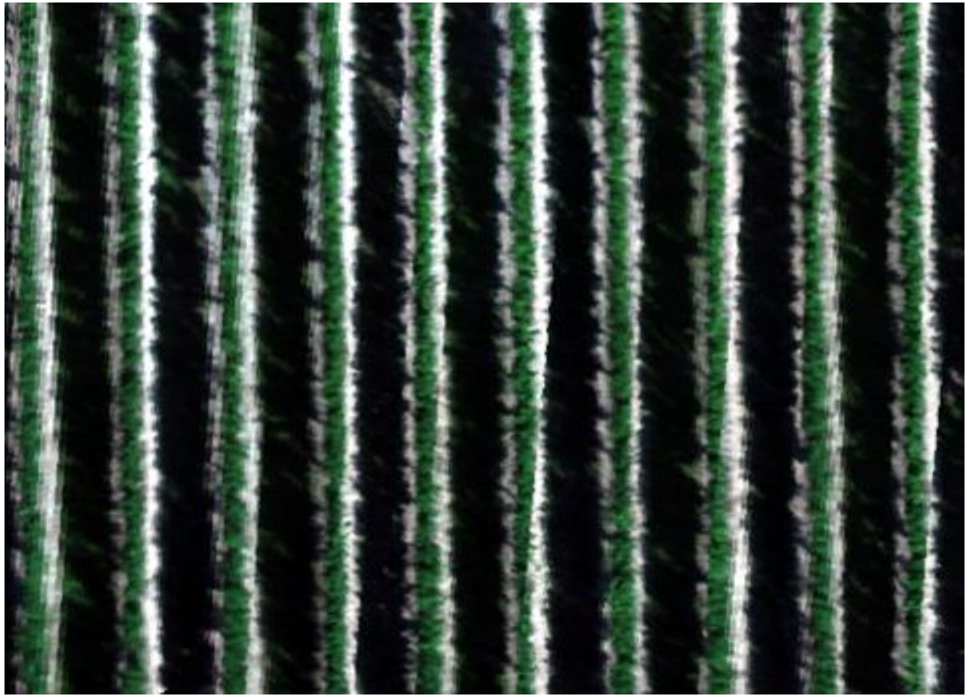
Chlorophyll content (R^2=0.87)

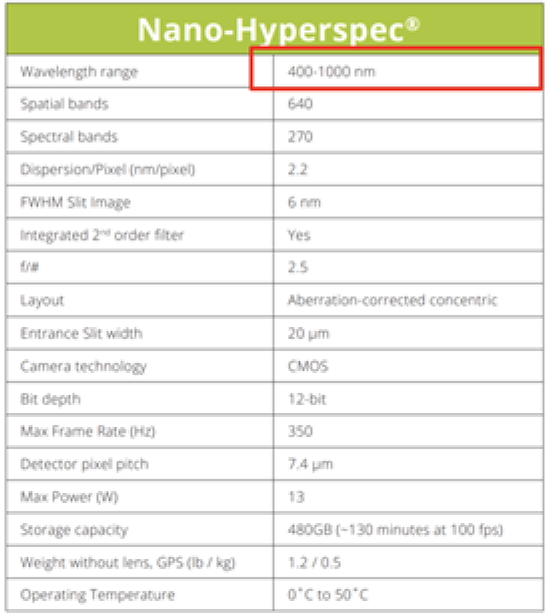


Leaf water thickness (R^2=0.83)

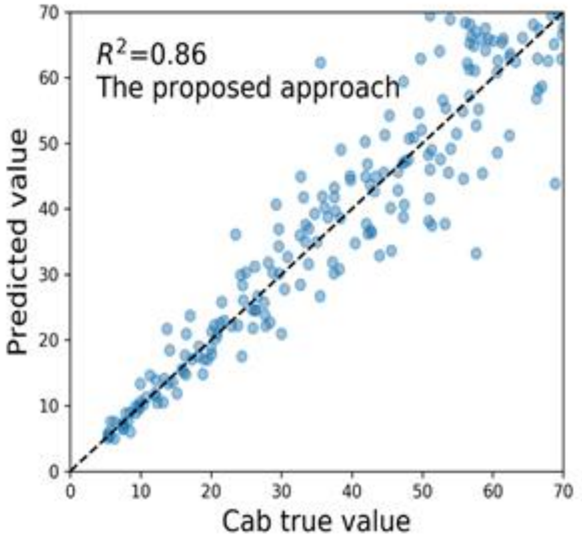
Headwall

Vineyard mapping using Headwall Pushbroom Camera

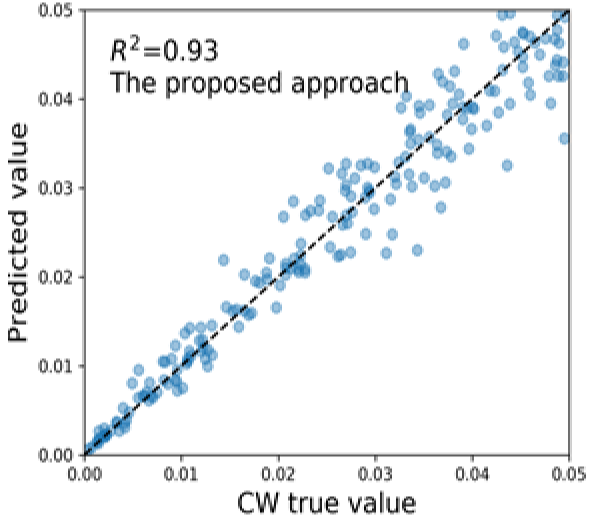

True-color RGB Composite Image



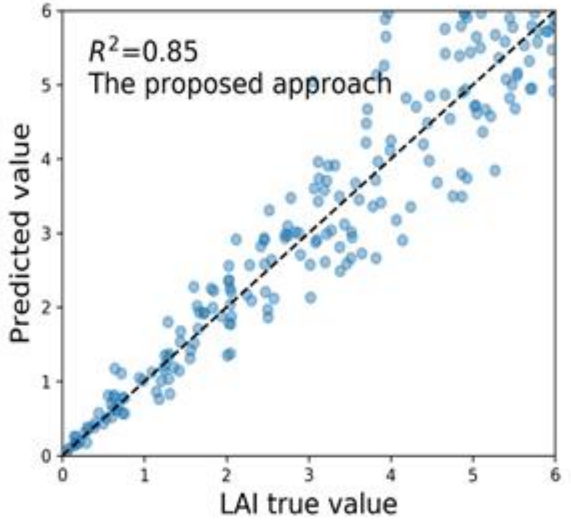
Nano-Hyperspec®	
Wavelength range	400-1000 nm
Spatial bands	640
Spectral bands	270
Dispersion/Pixel (nm/pixel)	2.2
FWHM Slit Image	6 nm
Integrated 2nd order filter	Yes
f/#	2.5
Layout	Aberration-corrected concentric
Entrance Slit width	20 μm
Camera technology	CMOS
Bit depth	12-bit
Max Frame Rate (Hz)	350
Detector pixel pitch	7.4 μm
Max Power (W)	13
Storage capacity	480GB (~130 minutes at 100 fps)
Weight without lens, GPS (lb / kg)	1.2 / 0.5
Operating Temperature	0°C to 50°C



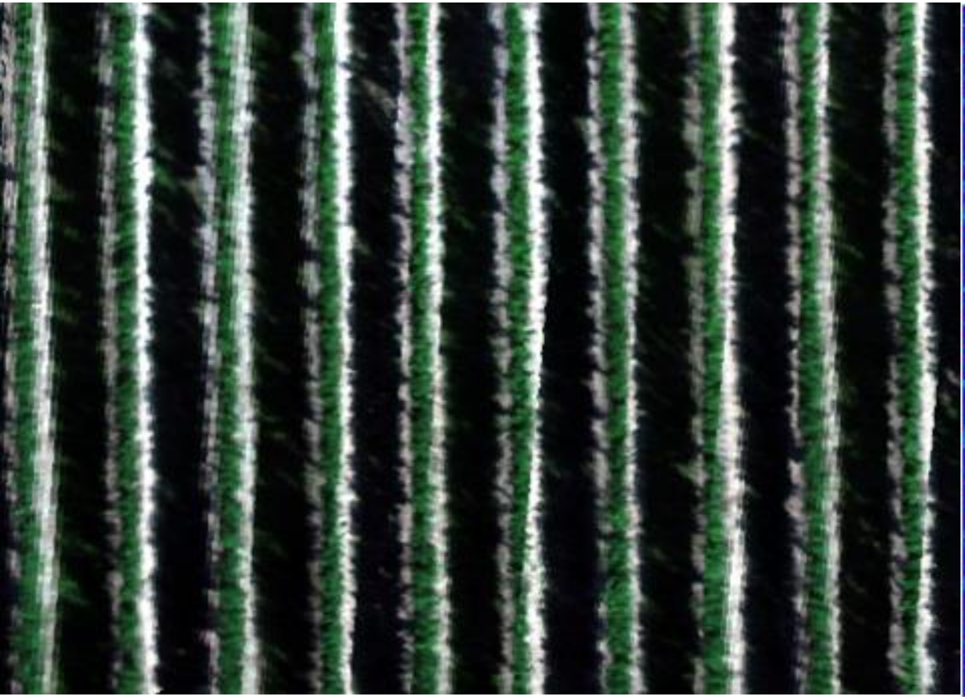
Result of ***Chlorophyll Concentration*** (Cab) Estimation

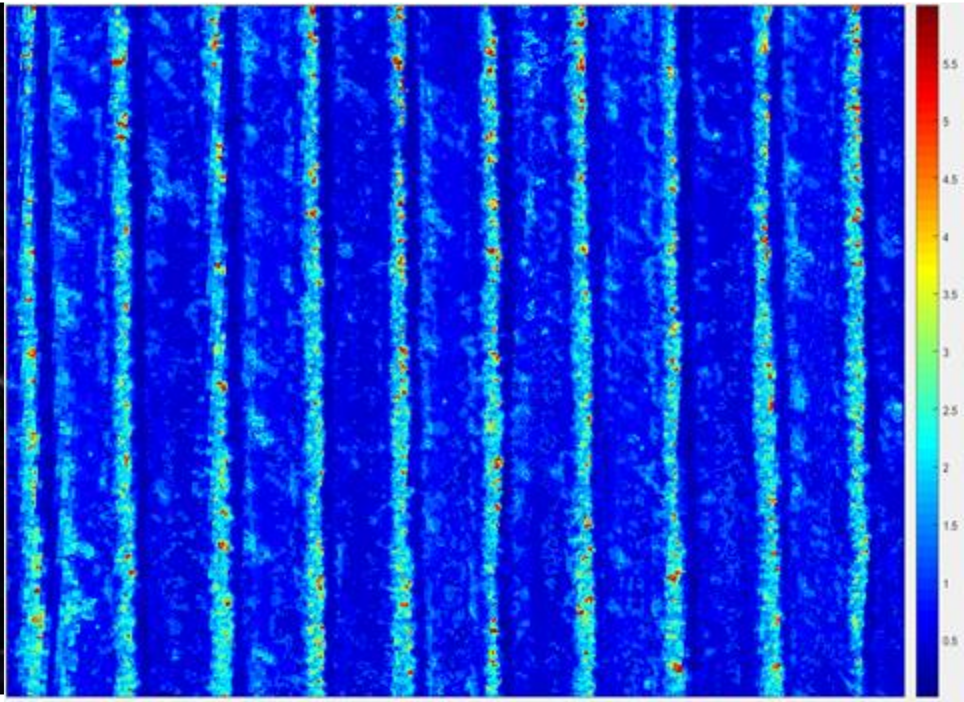


Result of ***Water Thickness*** (Cw) Estimation



Result of ***Leaf Area Index*** (LAI) Estimation

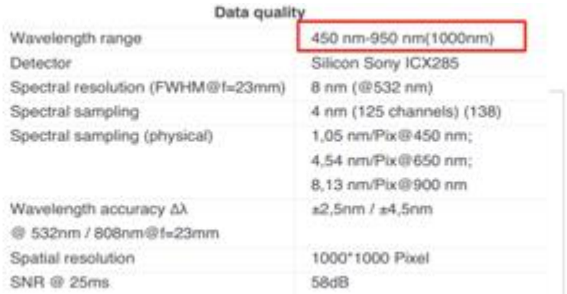

True-color RGB Composite Image



Estimated LAI

UAV Hyperspectral Imaging for Soil Iron Concentration Mapping

Hyperspectral camera: Cubert UHD 185-Firefly
Band range: 125 bands from ***450 to 950 nm***
Fight height: 178m
Images number: ***1804 images***
Image size 1000 × 1000 pixels
Spatial resolution: 0.05m
Data volume: ***241 GB***
# GT samples (mg/kg): 69

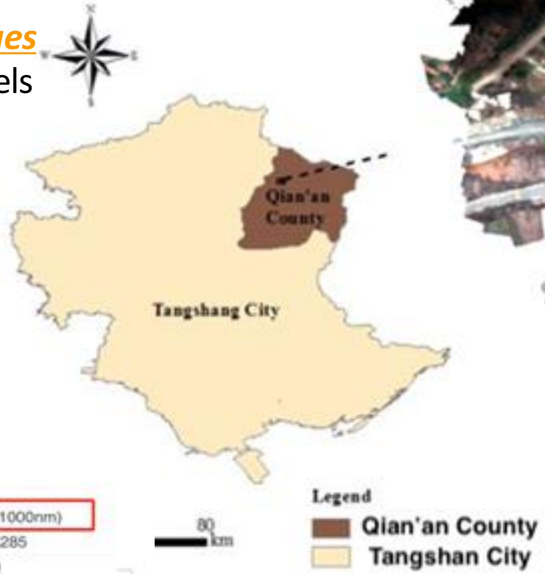


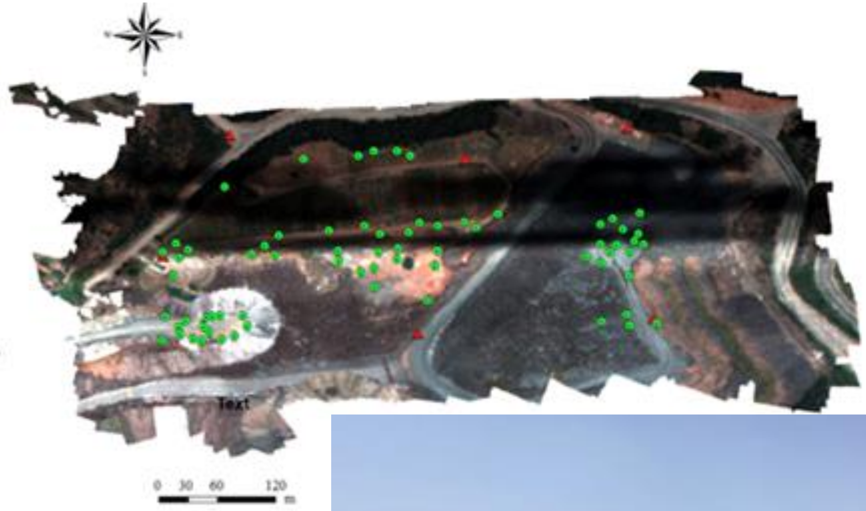
Data quality

Wavelength range	450 nm-950 nm(1000nm)
Detector	Silicon Sony ICX285
Spectral resolution (FWHM@f=23mm)	8 nm (@532 nm)
Spectral sampling	4 nm (125 channels) (138)

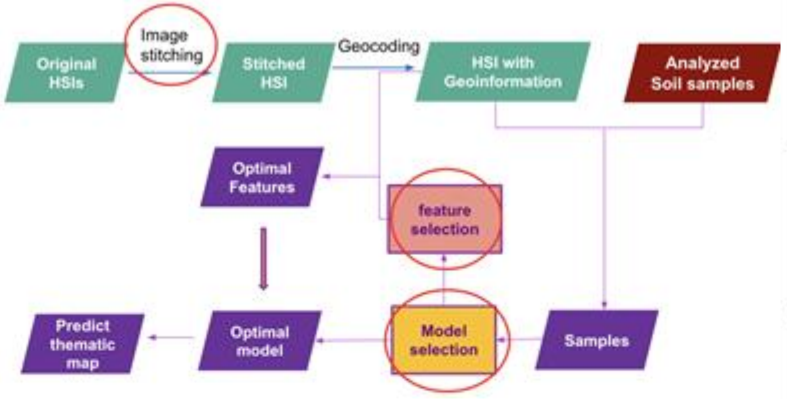
Spectral sampling (physical)	1,05 nm/Pix@450 nm;
	4,54 nm/Pix@650 nm;
	8,13 nm/Pix@900 nm

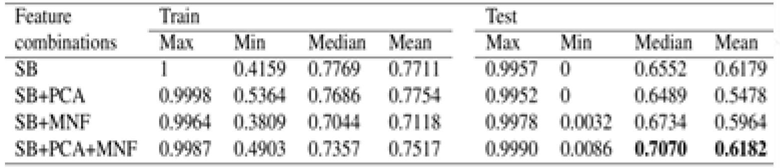
Wavelength accuracy Δλ @ 532nm / 808nm@f=23mm	±2,5nm / ±4,5nm
Spatial resolution	1000*1000 Pixel
SNR @ 25ms	58dB





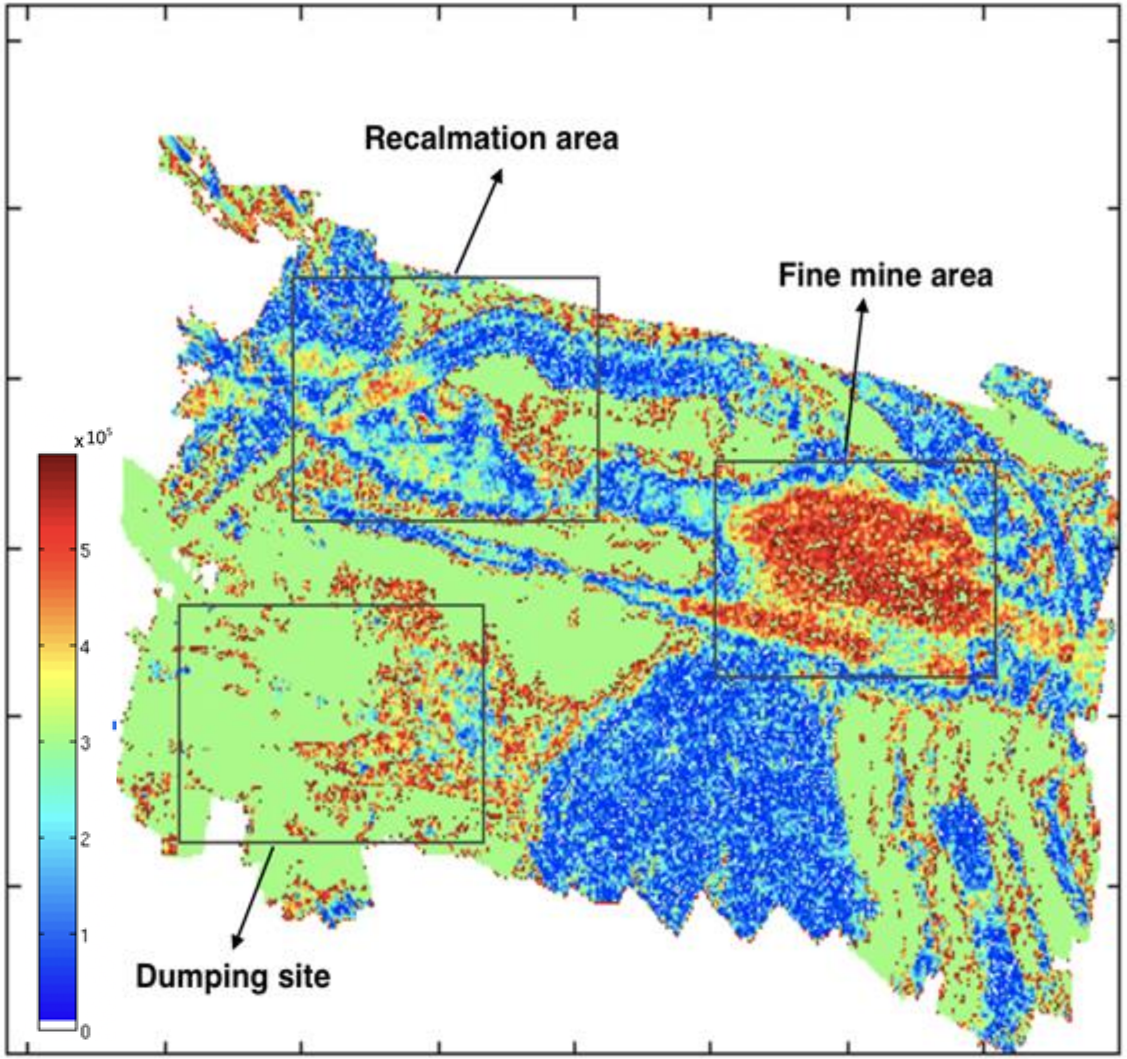
Hyperspectral soil mapping pipeline





Feature combinations	Train				Test			
	Max	Min	Median	Mean	Max	Min	Median	Mean
SB	1	0.4159	0.7769	0.7711	0.9957	0	0.6552	0.6179
SB+PCA	0.9998	0.5364	0.7686	0.7754	0.9952	0	0.6489	0.5478
SB+MNF	0.9964	0.3809	0.7044	0.7118	0.9978	0.0032	0.6734	0.5964
SB+PCA+MNF	0.9987	0.4903	0.7357	0.7517	0.9990	0.0086	**0.7070**	**0.6182**

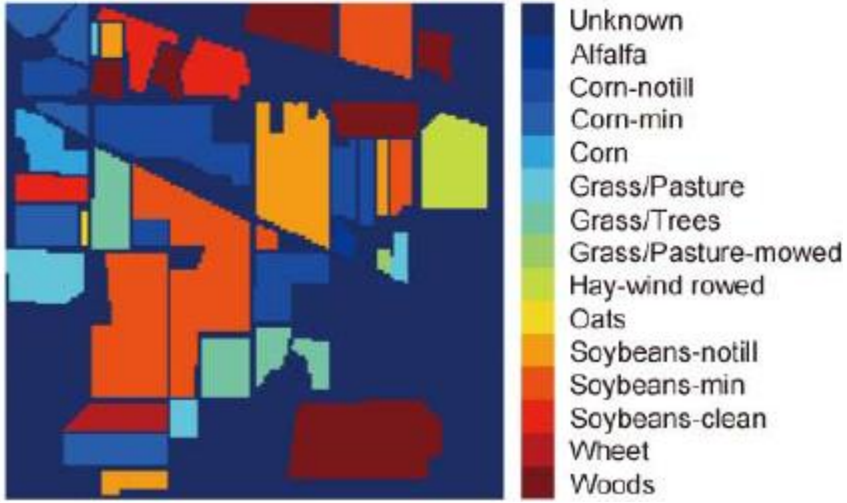
R^2 statistics obtained by different feature input using the PLSR model



Outline

Hyperspectral Image (HSI) Classification

RGB image Y



Classification map X

For each pixel in Y, we need to estimate its "identity", i.e., the semantic class membership

Key issues and challenges:

(1) different crops types have similar spectral pattern; weak spectral signature information -> requires efficient feature extraction methods;
(2) Which model is most suitable for this image? model selection;
(3) weak edges among classes; how to efficiently preserve edges in classification map?

Problem formulation

Forward model:

$Y = f(X)$

(1) Y: Remote sensing image, e.g., SAR, multispectral, hyperspectral images

(2) X: The “identity” or “class labels” of each pixel in Y

(3) ***f(.): The forward model, which is unknown***;

True inverse function:

$X = t(Y) = f^{-1}(Y)$

where ***t(.) is the true inverse function that is unknown, because the forward model is unknown;***

Data-driven approximated inverse function:

$X = g(Y)$

Note that g(.) is only an approximation to the true inverse function t(.), and g(.) is empirical model.

Based on $\{(X_j,Y_j) \mid j=1,2,...,n\}$, we build the following objective function:

$J(\theta) = \sum||X_i-g(Y_i)||$

$\theta = \min J(\theta)$

Model selection issue: how to achieve g(.) that approximate t(.) as close as possible?

Mixed Pixel in Hyperspectral Remote Sensing Image

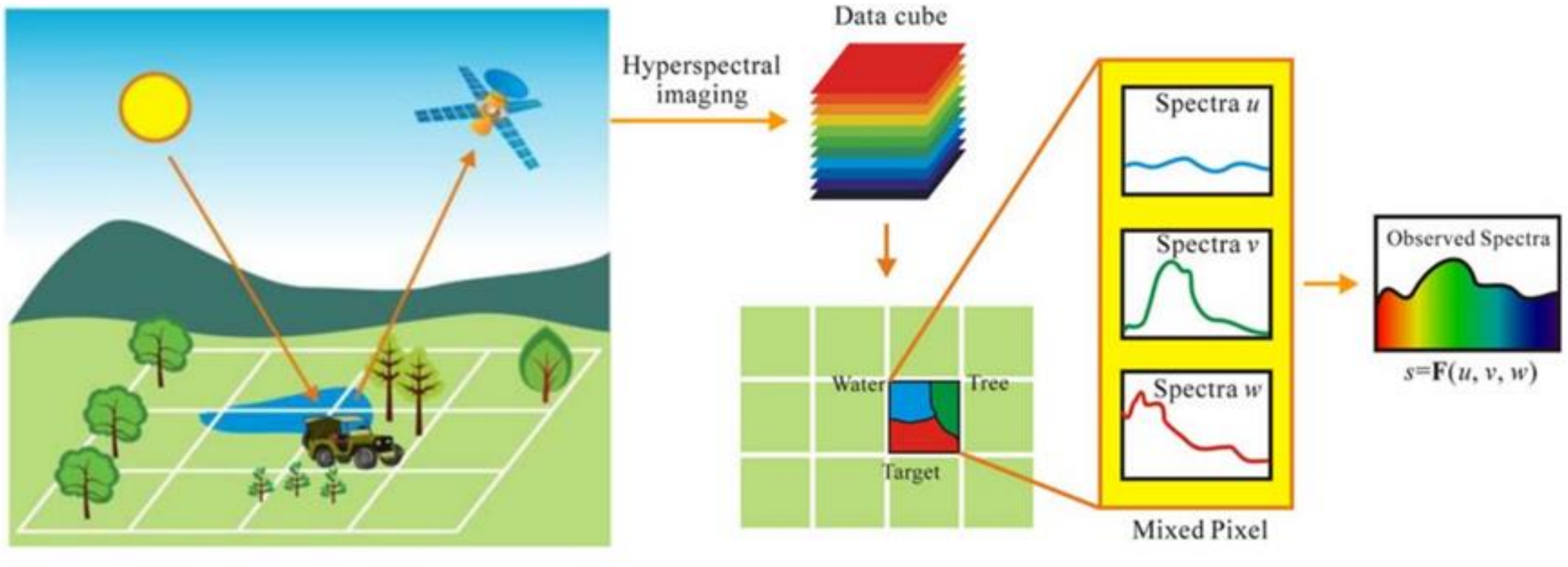
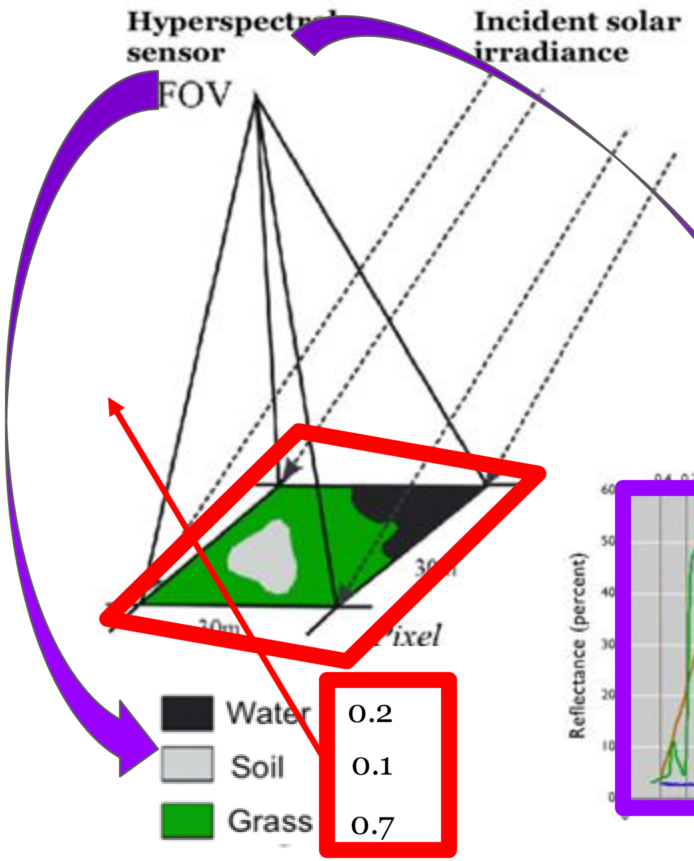


Illustration of mixed pixel generation in hyperspectral remote sensing (from Zhang et al. 2014)

Spectral Unmixing

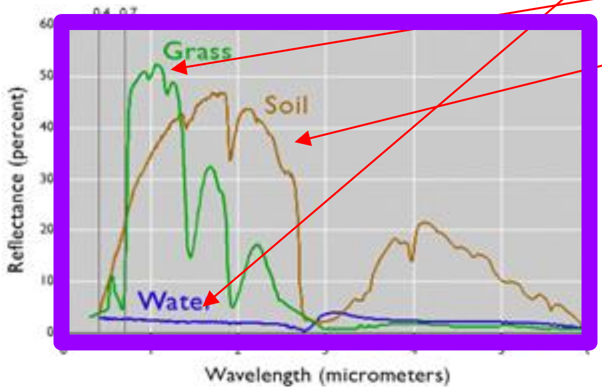


Within a 30m-by-30m IFOV, there are ***multiple land cover classes***, i.e., water, soil and grass.

The ***spatial heterogeneity*** of land cover classes leads to ***mixed pixels in HSI***.

The resulting spectra observation of this mixed pixel is a mixture of three pure spectra, i.e., grass, soil and water.

$$y_i = 0.2 \times a_{water} + 0.7 \times a_{grass} + 0.1 \times a_{soil}$$



Spectral unmixing aims to ***quantify the within-pixel spatial heterogeneity*** by decomposing the mixed pixel into pure spectra (i.e., ***endmembers***) and their fractional proportions (i.e., ***abundances***).

Spectral unmixing is essential for ***Discovering patterns using a limited number of abundances maps***

Spectral Unmixing

- **Spectral unmixing** aims to disentangle the mixed pixels y_i in hyperspectral image (HSI), and estimate both the **endmembers a_k** and the **abundance x_{ik}** simultaneously.

$$y_i = \sum_{k=1}^{K} a_k x_{ik} + n_i \quad (for\ \ i = 1, 2, ..., N)$$

The *i*th mixed pixel

The noise effect

The *k*th endmember

The abundance of *k*th endmember

- **Spectral unmixing** is **fundamental for quantitative information retrieval** from HSI, and is able to support various other HSI processing tasks, such as denoising, super-resolution, subpixel mapping and classification.

Classification vs. Spectral Unmixing

(1) Classification

- **Forward model:** $Y = f(X)$, where $f(.)$ does not exist.
- **True inverse function:** $X = t(Y) = f^{-1}(Y)$, where the true inverse function $t(.)$ is unknown
- **Approximated inverse function by supervised learning:** $X = g(Y)$

  Approximate $t(.)$ using raining pairs: $\{(X_j, Y_j) \mid j=1,2,\ldots,n\}$, -------> $J(\theta) = \sum||X_j\text{-}g(Y_j)||$, -----> $\theta = \min J(\theta)$

(1) Spectral Unmixing

- **Forward model:** $Y = f(X)=AX+N$, where $f(.)$ is the linear spectral mixture model
- **True inverse function:** $X = t(Y) = f^{-1}(Y)=A^{-1}(Y\text{-}N)$, however, A is generally not invertible (low rank)
- ***Estimate*** X_i **using constrained linear optimization:**

  Given (X_i, Y_i), -------> $J(X_i) = ||Y_i\text{-}AX_i||$, -----> $X_i = \min J(X_i)$

	(1) Direct inversion	(2) LUT approach	(3) Numerical Approach	(4) Simulation & ML	(5) ML	**(6) DL**
f(.) is known	yes	yes	yes	yes	yes	**yes**
f(.) is partially known, i.e., form known, but with some unknown parameters U	no	no	Yes, estimate X and U together	no	no	**no**
f(.) unknown, (X,Y) known	no	no	no	no	yes	**yes**
f(.) unknown, (X,Y) unknown	no	no	no	no	no	**no**
If both f(.) and (X,Y) known, can accommodate both?	no	yes?	Yes? Use (X,Y) to estimate parameters in f(.)	yes?	Yes, use both simulated and observed data	**Yes, use both simulated and observed data**
Can use prior information? e.g., spatial prior and value prior	no	Yes? Use value prior for sampling	Yes? Use value prior of X in Bayesian estimation	Yes, Use value prior in sampling and spatial prior in Random fields	Yes, spatial prior in Random field approaches	**Yes, similar to ML**
Advantages	Knowledge -driven; Simple, easy	Knowledge-driven; Intuitive, easy, discrete fitting;	Knowledge-driven; estimate U; Efficient for simple f(.) in convex problems	Knowledge-driven; flexible; continuous fitting; good inter/extrapolation; faster than LUT	Data-driven; flexible; Classic;	**Strong modeling capability; automatic feature learning;**
Disadvantages	Unrealistic; rely on simple f(.)	Sensitive to accuracy of f(.), similarity metrics, sampling density and range; slow if LUT is large; bad for extrapolation;	Rely on efficiency of nonlinear solver; Slow; Local optimum;	Overfitting and underfitting risk to simulated data; difficult model selection; Sensitive to accuracy of f(.), similarity metrics, sampling density and range;	Weak modeling capability; Rely on "good" engineered features; Black-box; Overfitting, underfitting; Feature and model selection is difficult and slow	**Overfitting and underfitting; Black-box;**

(3) Numerical Approaches

If the radiative transfer model f(.) is known, and we have an remote sensing observation Y, we can use the numerical approach to estimate the associated X.

Forward model: Y = f(X), where f(.) is the radiative transfer models, which tend to be highly nonlinear and un-invertible.

Based on some observations {Y}, we can build an objective function:

J(X) = Y-f(X)

X = min J(X)

We try to find X that through f(.) can generate output whose value is very close to the value of Y.

There are many methods that can solve this nonlinear optimization problems, for example,
---- Newton's method
---- Gradient descent methods
---- Simulated annealing approach

Because the forward model f(.) contains knowledge and physical rules, f(.) is usually called physical model.

Spectral Unmixing -- What if both *A* and *S* are unknown

Forward model: $Y = f(X)=AX+N$, where $f(.)$ is the linear spectral mixture model

True inverse function: $X = t(Y) = f^{-1}(Y)=A^{-1}(Y-N)$, however, A is generally not invertible (low rank)

- *Only X_i is unknown, estimate X_i using constrained linear optimization:*

  $X_i = min\ J(X_i)$, where $J(X_i) = ||Y_i-AX_i||$

- *Both A and X_i are unknown, estimate X_i and A iteratively using Expectation-maximization (EM) algorithm:*

  E-step: estimate X_i based on A and Y_i :

  $X_i = min\ J(X_i)$, where $J(X_i) = ||Y_i-AX_i||$

  M-step: estimate A based on X_i *and* Y_i :

  $A = min\ J(A)$, where $J(A) = ||Y_i-AX_i||$

  Repeat E-step and M-step until convergence;

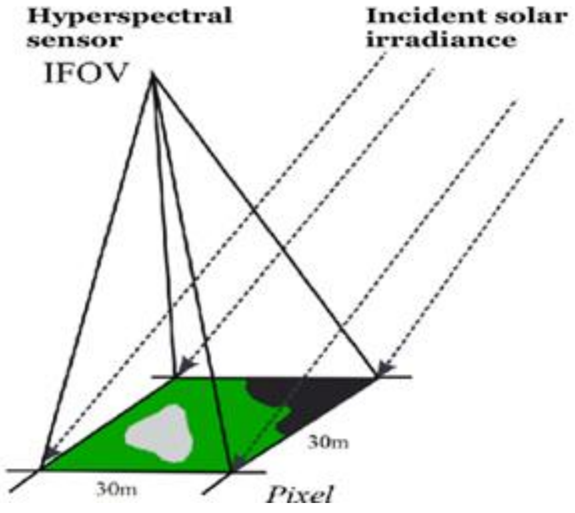
Outline

Linear spectral mixture model (LSMM)

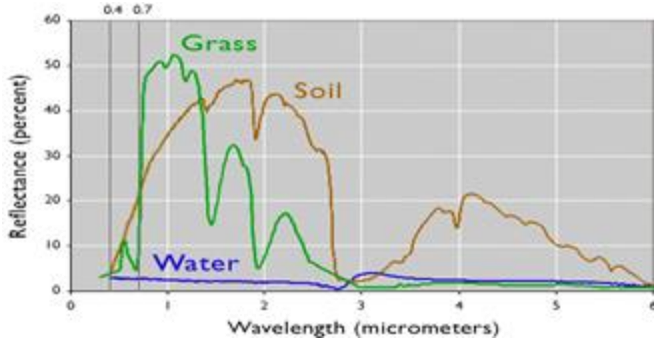
$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{S} + \boldsymbol{N}$$

$$\boldsymbol{x_i} = \sum_{k=1}^{K} \boldsymbol{a_k} s_i^k + \boldsymbol{n_i}$$

$$s_i^k > 0, \sum_k s_i^k = 1$$



⬛	Water	0.2
⬜	Soil	0.1
🟩	Grass	0.7



X: observation matrix of HSI
A: endmember matrix
S: abundance matrix

$$\boldsymbol{x} = 0.2 \times \boldsymbol{a}_{water} + 0.1 \times \boldsymbol{a}_{soil} + 0.7 \times \boldsymbol{a}_{grass}$$

How spectral unmixing helps other tasks?

- **HSI denoising:** achieved by reconstructing the “clean” pixel using estimated endmembers and abundances:

$$\hat{x}_i = \sum_{k=1}^{K} a_k s_i^k$$

- **HSI classification:** s_i can be treated as the soft label of pixels in HSI
- **HSI feature extraction**: achieved by estimating s_i from x_i .
- **Environmental monitoring:** Knowing the endmember-abundance pattern in HSI facilitates the qualification of the ground environment from HSI (such as fire burn severity, deforestation level and soil contamination).

Key issues of spectral unmixing

$$x_i = \sum_{k=1}^{K} a_k s_i^k + n_i$$

(1) The characterization of noise *n* in HSI

--- Over or under characterization of noise $\boldsymbol{n}$ cause inaccurate $\{\boldsymbol{s}_i\}$ and $\{\boldsymbol{a}_k\}$.

(2) The development of effective constraint on endmembers $\{\boldsymbol{a}_k\}$

--- Effective constraints on $\{\boldsymbol{a}_k\}$ serve as guidance and regulations during the estimation process.

(3) The modeling of abundance $\{\boldsymbol{s}_i\}$ in HSI

--- Accurate regulating and estimating of $\{\boldsymbol{s}_i\}$ relies on well leveraging the spatial contexture information.

(4) The design of fast and efficient model optimization techniques

Inverse problem optimization

Forward model: $\boldsymbol{x}_i = \boldsymbol{a}_k \boldsymbol{s}_i + \boldsymbol{n}_i$

Px1 PxK Kx1

Inverse model: $\{\boldsymbol{a}_k, \boldsymbol{s}_i\} = \Phi^{-1}(\boldsymbol{x}_i)$

$\boldsymbol{X} = \{\boldsymbol{x}_i | i = 1, 2, ..., M\} \in \mathbb{R}^{P \times M}$

$\boldsymbol{A} = \{\boldsymbol{a}_k | k = 1, ..., K\} \in \mathbb{R}^{P \times K}$

$\boldsymbol{S} = \{\boldsymbol{s}_i | i = 1, ..., M\} \in \mathbb{R}^{K \times M}$

Check the posedness:

- ❑ EXISTENCE (✓)
- ❑ UNIQUENESS (✗)
- ❑ CONTINUITY (✗)

Unknowns are much more than knowns.

- ☐ Infinite solutions
- ☐ <u>Uniqueness fails</u>.

▯ The problem is ill-posed

▯ Prior knowledge is required

▯ Regulations

▯ Bayesian approach

Inverse problem optimization

Maximum a posteriori (MAP):

$$\{\hat{\boldsymbol{A}}, \hat{\boldsymbol{S}}\} = arg \max_{\boldsymbol{A}.\boldsymbol{S}} \{p(\boldsymbol{A}, \boldsymbol{S}|\boldsymbol{X})\}$$

Posterior distribution:

$$p(\boldsymbol{A}, \boldsymbol{S}|\boldsymbol{X}) \propto p(\boldsymbol{X}|\boldsymbol{A}, \boldsymbol{S}) p(\boldsymbol{A}|\boldsymbol{S}) p(\boldsymbol{S}$$

Objective function:

$$\begin{aligned} J_{A,S} &= arg \min_{\boldsymbol{A},\boldsymbol{S}} \{-logp(\boldsymbol{A}, \boldsymbol{S}|\boldsymbol{X})\} \\ &\propto arg \min_{\boldsymbol{A}.\boldsymbol{S}} \{-logp(\boldsymbol{X}|\boldsymbol{A}, \boldsymbol{S}) - logp(\boldsymbol{S}) - logp(\boldsymbol{A}|\boldsymbol{S})\} \end{aligned}$$

Key research issues

$$p(A, S|X) \propto p(X|A, S)p(A|S)p(S$$

1) The modelling of the data likelihood $p(\boldsymbol{X}|\boldsymbol{A}, \boldsymbol{S})$.

2) The modelling of the conditional distribution $p(\boldsymbol{A}|\boldsymbol{S})$.

3) The accurate modelling of the abundance prior $p(\boldsymbol{S})$.

4) The design of an efficient optimization scheme for solving the MAP problem.

$$x_i = \sum_{k=1}^{K} a_k s_i^k + n_i$$

1) The characterization of noise n in HSI.

2) The development of effective constraint on endmembers $\{a_k\}$.

3) The modeling of abundance $\{s_i\}$ in HSI.

4) The design of fast and efficient model optimization techniques.

1. Introduction
2. Hyperspectral unmixing
3. **Key research issues**
   - **Characterization of noise in HSI**
   - **Endmember constraints**
   - **Modelling the spatial correlation in abundances**
4. Implementation of a Bayesian spectral unmixing framework

Characterization of noise in HSI

- **Thermal noise and quantization noise** are signal independent and usually Gaussian distributed.
- Other noise types: shot noise, sparse noise, pattern noise

- Current imaging systems that are designed based on the assumption of **additive Gaussian noise** perform quite well.

- **Noise levels** of HSI **vary** dramatically **over bands** for most sensors due to different spectral absorption properties of different spectral channels and the typical existence of "junk bands".

Noise variance heterogeneity

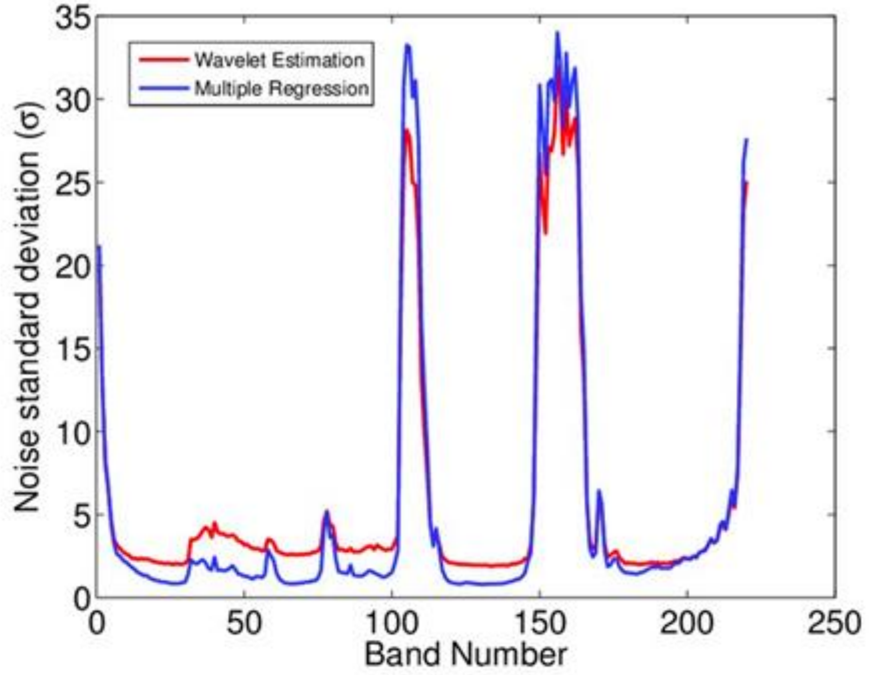


Figure: Comparison of noise estimation of **Indian Pines** in the wavelet domain and by using multiple regression approach.

Rasti, Behnood. *Sparse hyperspectral image modeling and restoration*. Diss. Ph. D. dissertation, Dept. Elect. Comput. Eng., Univ. Iceland, Reykjavik, Iceland, 2014.

Characterization of noise in HSI

Current SU methods:
assume that noise in different bands are IID Gaussian noise.
--- undesirable preservation of noise in some bands & erasing of the signal in some other bands.

- ➢ The modelling of the noise variance heterogeneity effect in HSIs.

Constraints on endmembers

- Geometrical-based algorithms
- Prior distribution constraints in the Bayesian framework
- K-P-means
- Endmember variability modelling
  - The purified means constraint on $\boldsymbol{A}$ for SU.
  - The modelling of the endmember variability effect.
  - The selection of the proper prior distribution of $\boldsymbol{A}$.

Modelling of large-scale non-stationary spatial correlation in abundances

- ❑ Graphical models e.g., conditional random field(CRF)
- ❑ Non-local approaches e.g., Non-local networks
- ❑ Deep image prior （DIP）

Traditional SU method (NNLS):
-- ignore the spatial correlation effect of $\{s_i^k\}$.
-- small scale and isotropic correlations using MRF stationary spatial correlation.

Jasper Ridge

- ➢ Leveraging DIP using a FCNN for abundance mapping.
- ➢ Applying the CRF approach on S as a post-processing method.
- ➢ Incorporating non-local neural network to CNN or FCNN.

1. Introduction
2. Hyperspectral unmixing
3. Key research issues
   - Characterization of noise in HSI
   - Endmember constraints
   - Modelling spatial correlation in abundances
4. **Implementation of a Bayesian spectral unmixing framework**

Key research issues

$$J_{A,S} = arg \min_{A,S}\{-logp(\boldsymbol{A}, \boldsymbol{S}|\boldsymbol{X})\}$$
$$\propto arg \min_{A,S}\{-logp(\boldsymbol{X}|\boldsymbol{A}, \boldsymbol{S}) - logp(\boldsymbol{S}) - logp(\boldsymbol{A}|\boldsymbol{S})\}$$

$$\boldsymbol{x}_i = \sum_{k=1}^{K} \boldsymbol{a}_k s_i^k + \boldsymbol{n}_i$$

1) The modelling of the data likelihood $p(\boldsymbol{X}|\boldsymbol{A}, \boldsymbol{S})$.

2) The modelling of the conditional distribution $p(\boldsymbol{A}|\boldsymbol{S})$.

3) The accurate modelling of the abundance prior $p(\boldsymbol{S})$.

4) The design of an efficient optimization scheme for solving the MAP problem.

1) The characterization of noise n in HSI.

2) The development of effective constraint on endmembers $\{\boldsymbol{a}_k\}$.

3) The modeling of abundance $\{s_i\}$ in HSI.

4) The design of fast and efficient model optimization techniques.

Data likelihood with heterogeneous noise variance -- $p(\boldsymbol{X}|\boldsymbol{A}, \boldsymbol{S})$

$$\boldsymbol{X} \xrightarrow{\text{Spectral unmixing}} \hat{\boldsymbol{A}}, \hat{\boldsymbol{S}} \quad \xrightarrow[\boldsymbol{X} = \boldsymbol{AS} + \boldsymbol{N}]{\text{Forward model}} \hat{\boldsymbol{X}}$$

-- Noise variance heterogeneity

$$p(\boldsymbol{n}_i) = \frac{1}{\sqrt{(2\pi)^P |\boldsymbol{\Lambda}|}} exp(-\frac{1}{2}\boldsymbol{n}_i{}^T \boldsymbol{\Lambda}^{-1} \boldsymbol{n}_i)$$

$$\boldsymbol{\Lambda} = \begin{bmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_P^2 \end{bmatrix}$$

$$p(\boldsymbol{X}|\boldsymbol{A}, \boldsymbol{S}) = \prod_{i=1}^{M} \frac{1}{\sqrt{(2\pi)^P |\boldsymbol{\Lambda}|}} exp(-\frac{1}{2}(\boldsymbol{X} - \boldsymbol{AS})^T \boldsymbol{\Lambda}^{-1} (\boldsymbol{X} - \boldsymbol{AS}))$$

Conditional distribution of endmembers given abundance with purified means -- $p(\boldsymbol{A}|\boldsymbol{S})$

$$p(\boldsymbol{a}_k|\boldsymbol{S},\boldsymbol{a}_{j\neq k}) = \frac{1}{z} exp(-||\underline{\boldsymbol{a}_k - E(\boldsymbol{a}_k|\boldsymbol{S},\boldsymbol{a}_{j\neq k})}||^2)$$

$$p(\boldsymbol{A}|\boldsymbol{S}) = \prod_{i=1}^{K} p(\boldsymbol{a}_k|\boldsymbol{S},\boldsymbol{a}_{j\neq k}$$

Achieved by the endmember extraction algorithm "K-P-Means"

$$\boxed{\boldsymbol{y}_i^k = (\boldsymbol{x}_i^k - \sum_{j\neq k}^{K} s_i^j \boldsymbol{a}_j)/s_i^k}$$

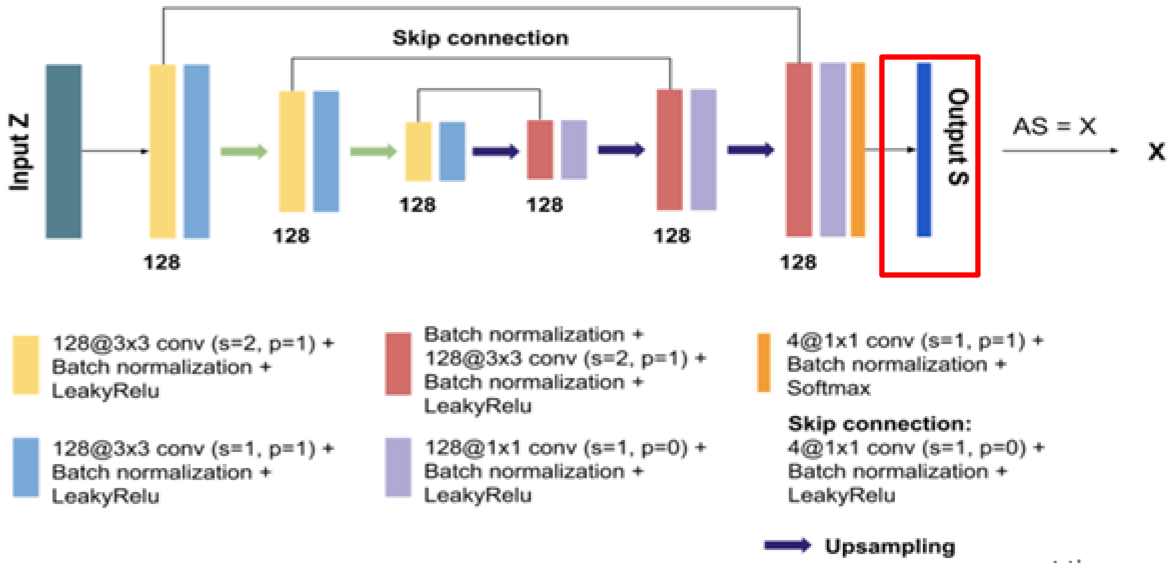
$$\boxed{\hat{\boldsymbol{a}}_k = \frac{1}{M}\sum_{i}^{M} \boldsymbol{y}_i^k}$$

Prior of abundance with DIP – $p(\boldsymbol{S})$

$$p(\boldsymbol{S}) = \frac{1}{z} exp(-||\boldsymbol{S} - E(\boldsymbol{S})||^{2}$$

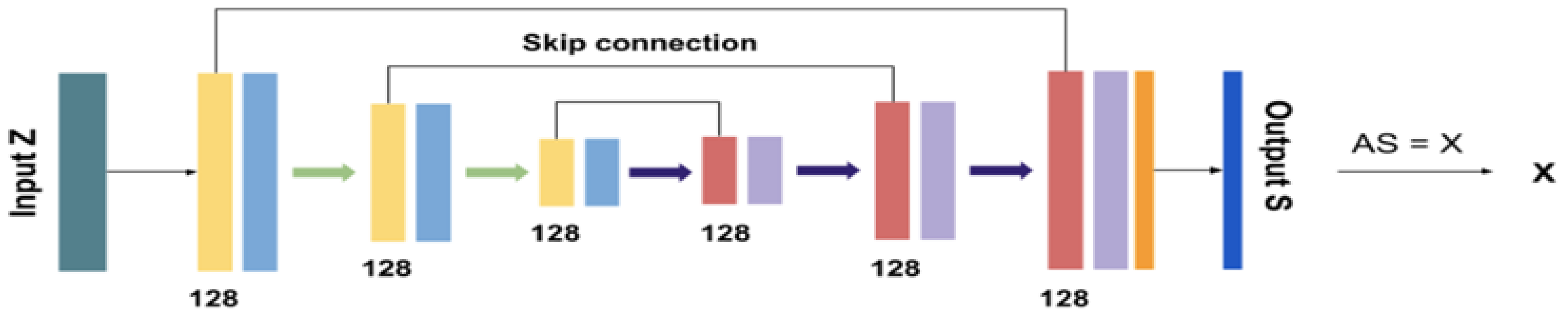
$$E(\boldsymbol{S}) = f(\boldsymbol{Z}, \beta$$

$f(\cdot)$: The forward propagation of fully convolutional neural network (FCNN).


Skip connection
Input Z
128
128
128
128
128
128
Output S
AS = X
X
128@3x3 conv (s=2, p=1) + Batch normalization + LeakyRelu
128@3x3 conv (s=1, p=1) + Batch normalization + LeakyRelu
Batch normalization + 128@3x3 conv (s=2, p=1) + Batch normalization + LeakyRelu
128@1x1 conv (s=1, p=0) + Batch normalization + LeakyRelu
4@1x1 conv (s=1, p=1) + Batch normalization + Softmax
Skip connection: 4@1x1 conv (s=1, p=0) + Batch normalization + LeakyRelu
Upsampling

Ulyanov, Dmitry, Andrea Vedaldi, and Victor Lempitsky. "Deep image prior." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

The proposed Bayesian convolutional unmixing network (BCUN)



🟨 128@3x3 conv (s=2, p=1) + Batch normalization + LeakyRelu

🟦 128@3x3 conv (s=1, p=1) + Batch normalization + LeakyRelu

🟥 Batch normalization + 128@3x3 conv (s=2, p=1) + Batch normalization + LeakyRelu

🟪 128@1x1 conv (s=1, p=0) + Batch normalization + LeakyRelu

🟧 4@1x1 conv (s=1, p=1) + Batch normalization + Softmax

Skip connection:
4@1x1 conv (s=1, p=0) + Batch normalization + LeakyRelu

➡ **Upsampling**

Model optimization

-- Maximum a posteriori (MAP) optimization

$$J_{A,S} = arg \min_{\boldsymbol{A},\boldsymbol{S}} \{ ((\boldsymbol{X} - \boldsymbol{A}E(\boldsymbol{S}|\boldsymbol{X}))^T \boldsymbol{\Lambda}^{-1} (\boldsymbol{X} - \boldsymbol{A}E(\boldsymbol{S}|\boldsymbol{X}))) + \alpha \sum_{k=1}^{K} ||\boldsymbol{a}_k - E(\boldsymbol{a}_k|\boldsymbol{S}, \boldsymbol{a}_{j \neq k})||^2 \}$$

-- Expectation-Maximization (EM) algorithm

E-step: Given endmembers ***A*** estimate abundances ***S*** by optimizing a FCNN.

M-step: Given ***S***, estimate endmembers ***A***. Endmembers A are estimated using purified means approach.

Experiment Design

❖ **Dataset:**

Simulated HSI & real HSI (Jasper Ridge)

❖ **Methods compared:**

PPI, N-FINDR, VCA, Kpmeans, uDAs & BCUN

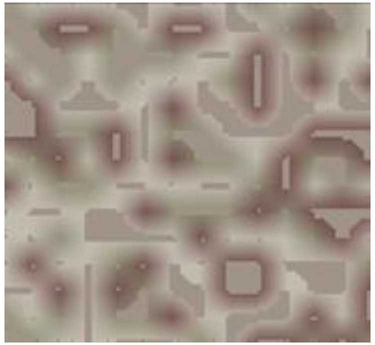
❖ **Numerical measure:**

Spectral angle distance (SAD)

Abundance angle distance (AAD)

Structural similarity (SSIM)

Mean squared error (MSE)


Simulated HSI
104x104x200

Real HSI
512x614x224

Test on Simulated HSI

Model comparison -- Abundance estimation

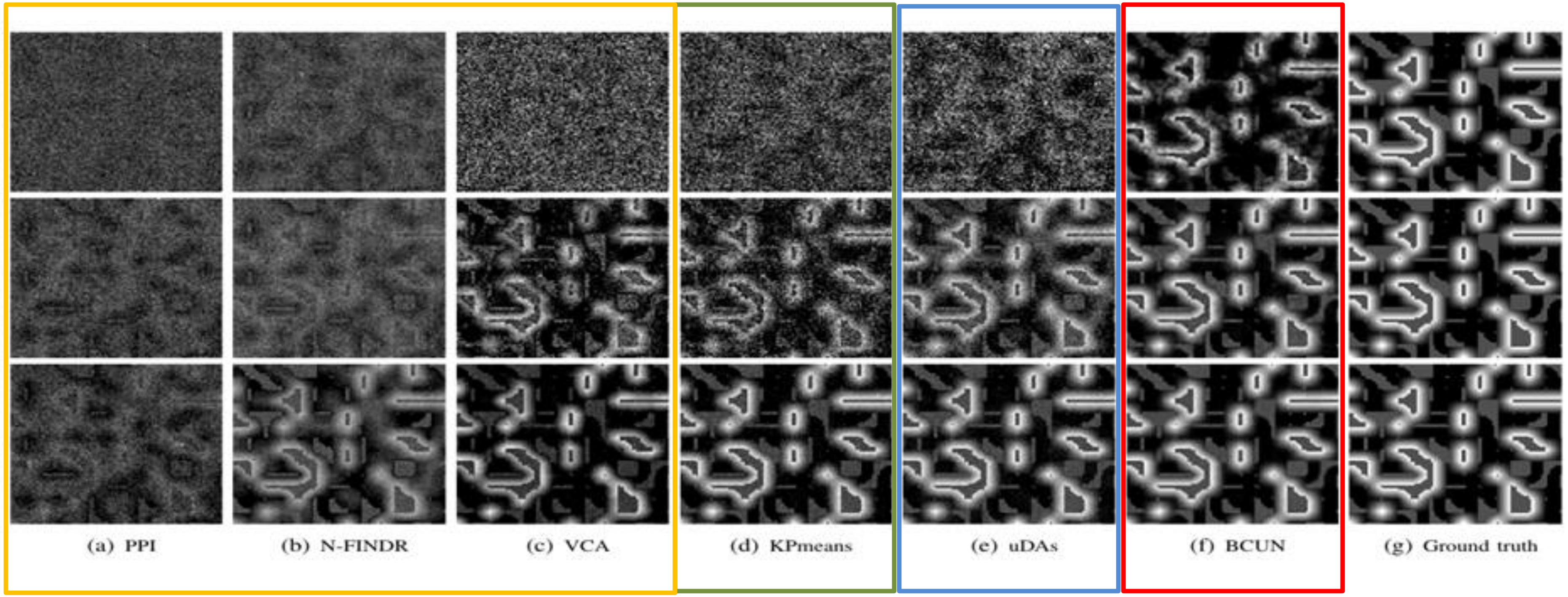


Figure 1. The abundance maps achieved by different methods on one endmember with different SNR values, i.e., 10, 20, 30dB from top row to bottom row and the ground truth (GT) at the last column.

Model comparison -- Endmember extraction

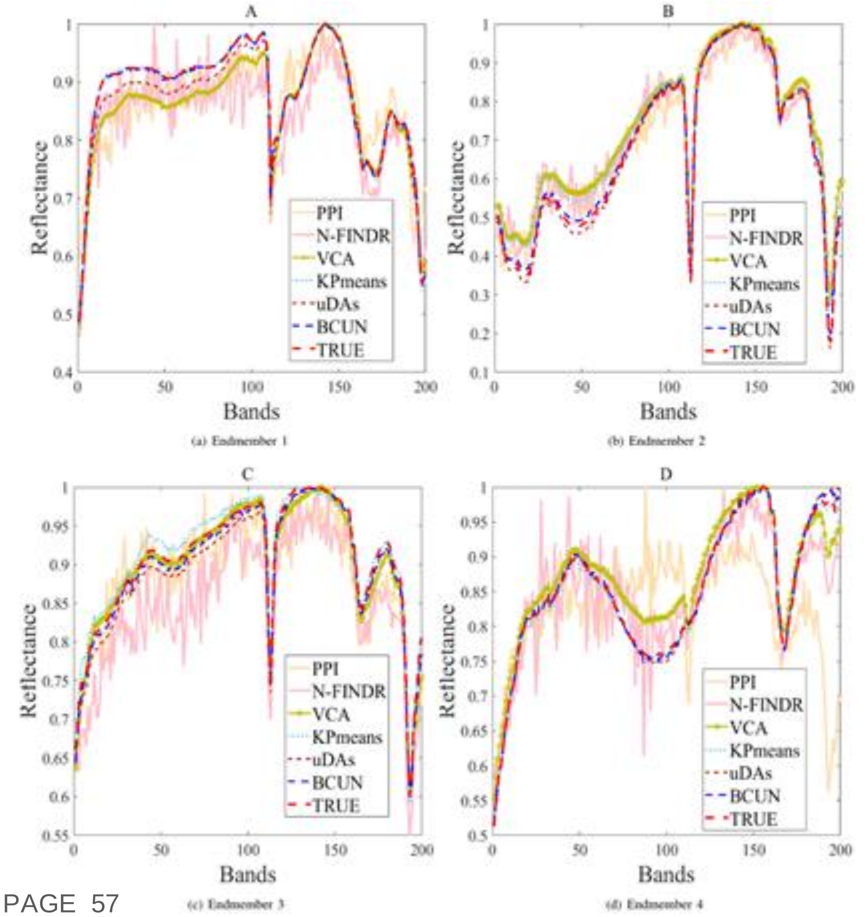


Figure 2. Four endmembers achieved by different methods on the HSI with SNR equals 30dB.

Test on real HSI

Model comparison -- Abundance estimation

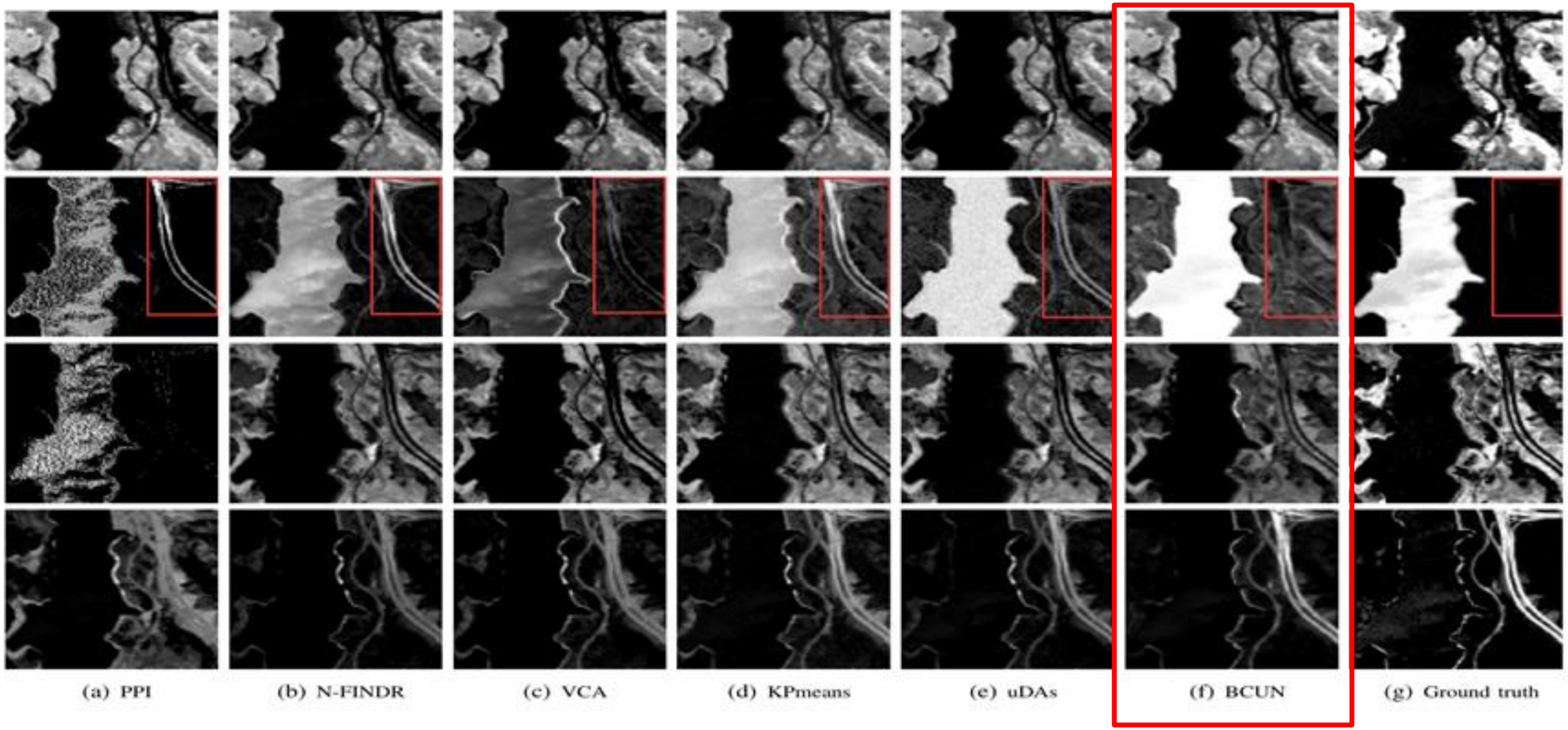


Figure 3. The abundance maps achieved by different methods on four endmembers (tree, water, soil, road) respectively from top row to bottom row.

Model comparison -- Endmember extraction

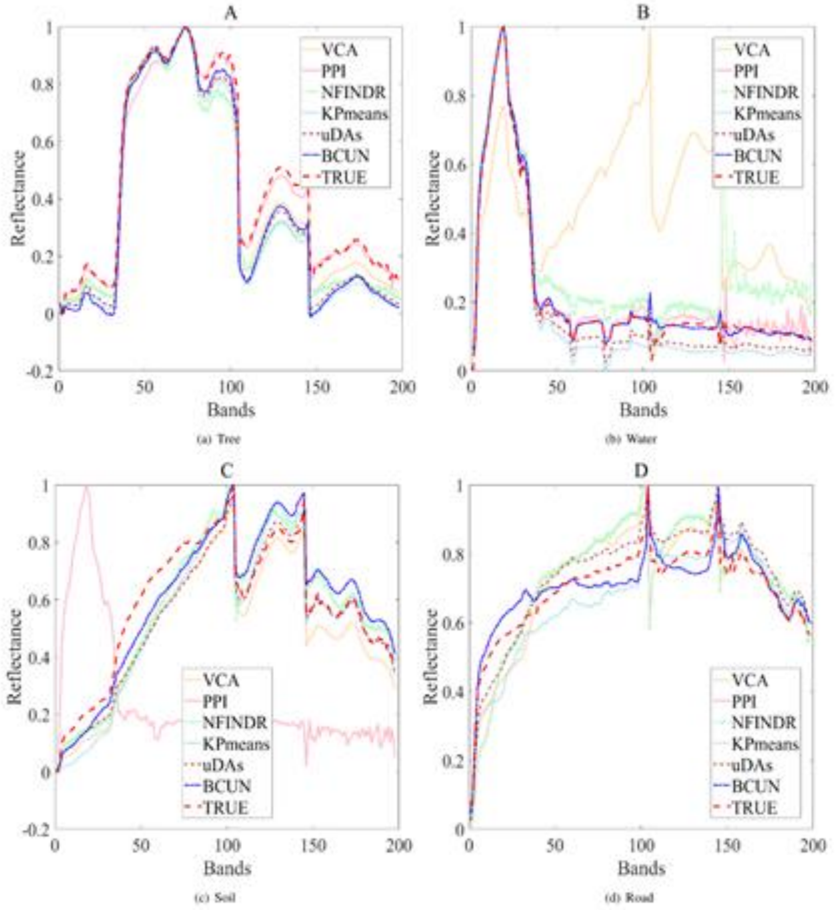


Figure 4. The estimated endmembers achieved by different methods, along with the references from the USGS library on four images about four materials separately from left column to right column.

Conclusion of experiments

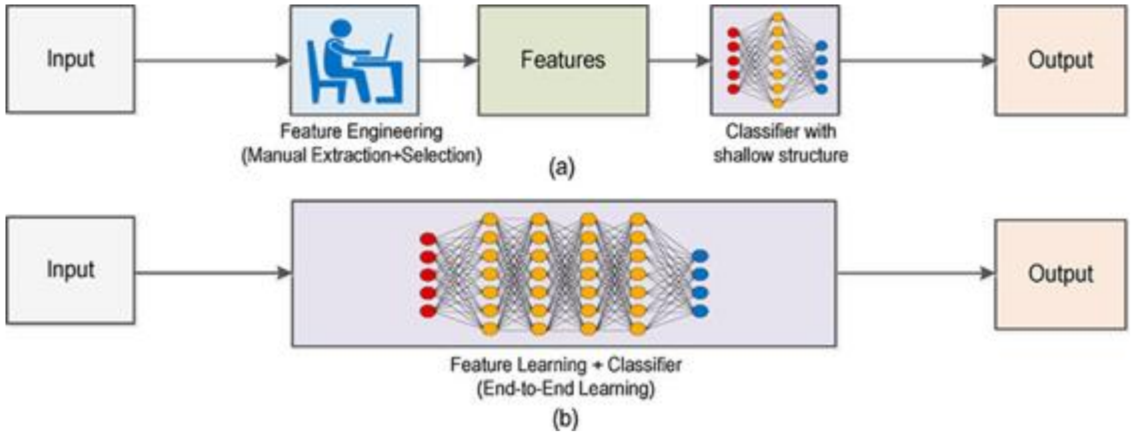
- The proposed BCUN approach constitutes a complete Bayesian approach **with effective modelling and optimization approaches** for enhanced spectral unmixing.

- The proposed approach was tested on both real and simulated HSI, in comparison with several other popular SU methods, and results demonstrated that the **proposed BCUN** method was more capable of **accurately estimating both the endmember and abundance** from HSIs.

Outline

- ➔ Hyperspectral Basics
- ➔ Hyperspectral Image Processing Tasks
- ➔ UAV Hyperspectral Crop and Soil Mapping
- ➔ Classification vs. Spectral Unmixing
- ➔ Hyperspectral Unmixing
- ➔ Hyperspectral Image Classification

HSI Classification
Knowledge-driven feature engineering vs. Data-driven deep learning (DL)



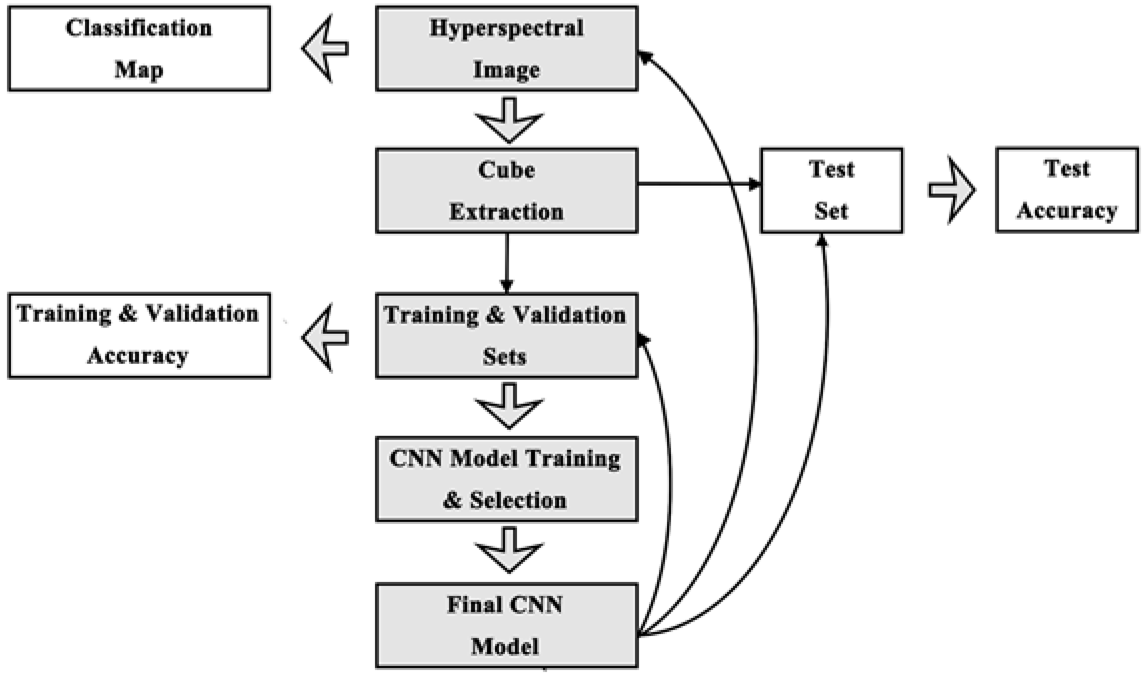
Advantages of DL approaches for RS image classification:

(1) automatically learn the "best" feature without requiring task-specific classifier-specific knowledge;

(2) End-to-end approach without any intermediate stages in the data-processing pipeline;

(3) Complex model -> strong modeling capability -> efficiently capture the subtle differences among classes;

(4) Powerful GPU computation



Flowchart of using CNN for HSI classification

Steps for Hyperspectral image (HSI) classification using the CNN approach:

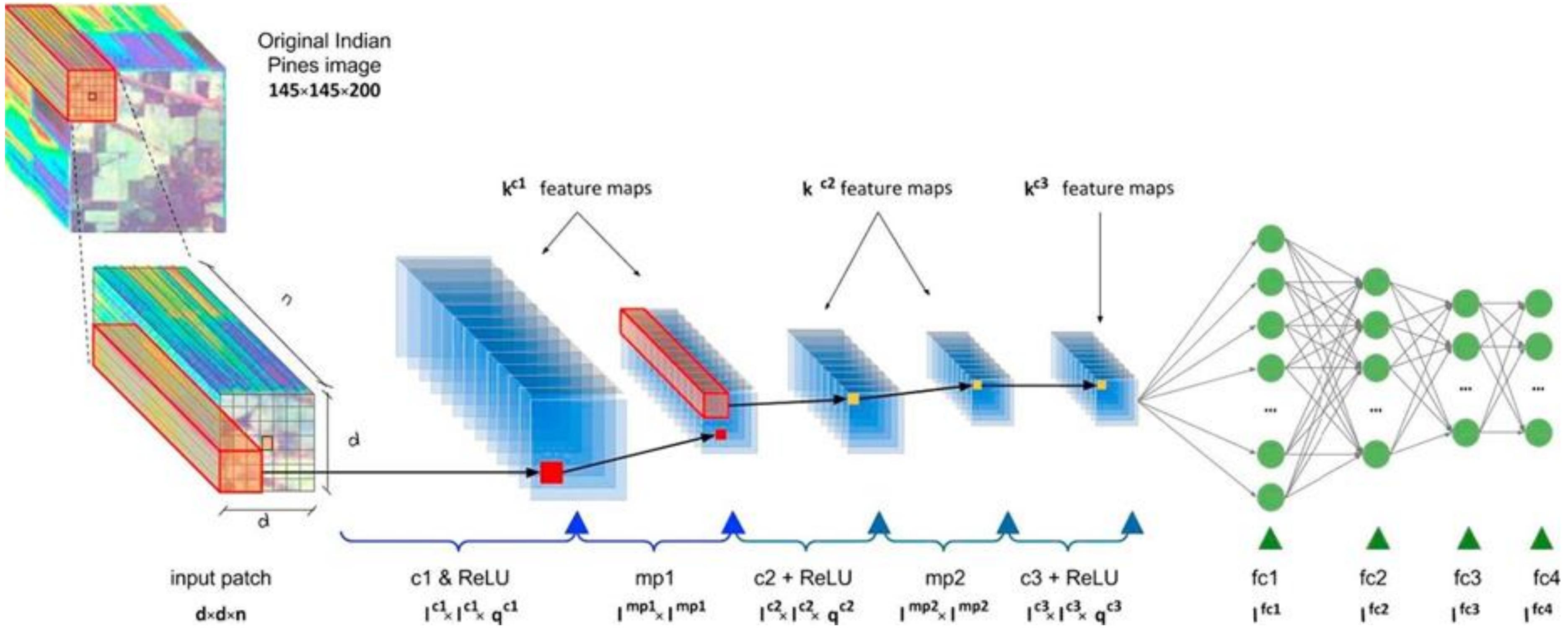
Step 1: 3D cube extraction, for each pixel with known label, extract a 3D cube centered at this pixel to use as Y_i;

Step 2: split the 3D cubes into **training set, validation set and test set**;

Step 3: train CNN on the training set, compare models using validation set and determine the "best" model architecture;

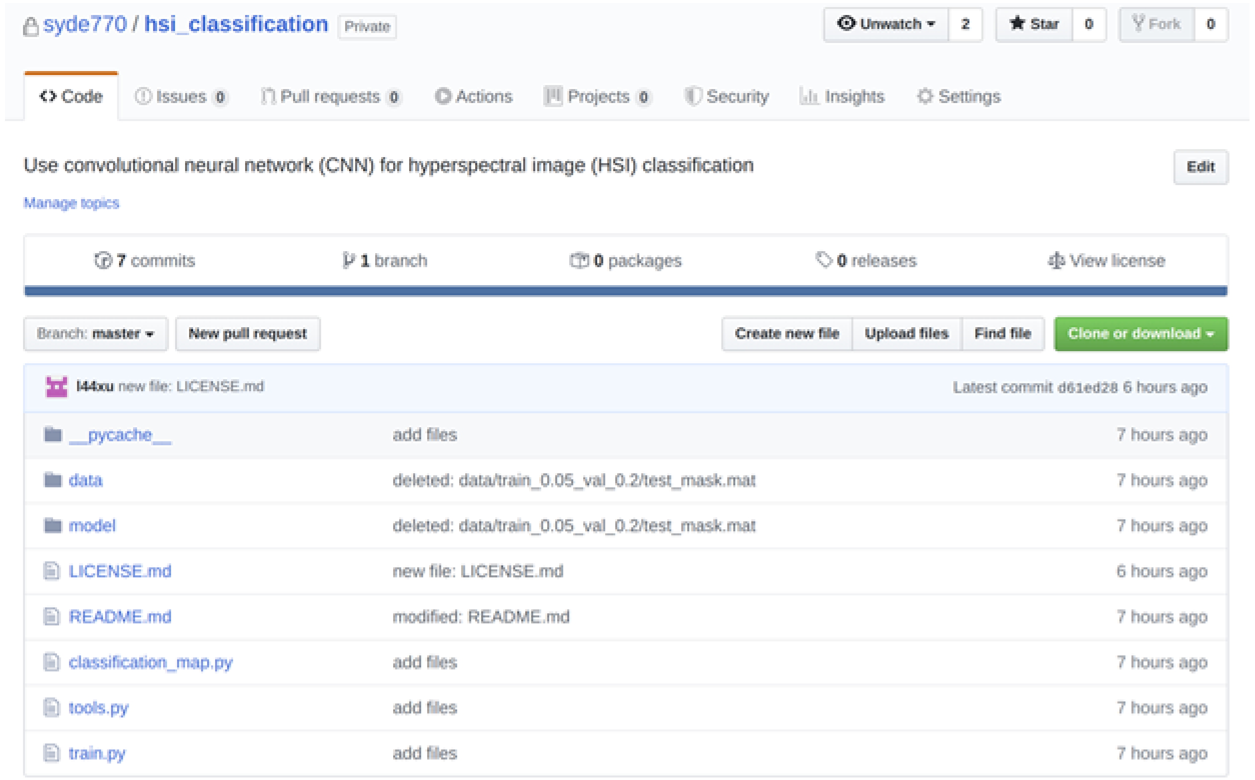
Step 4: generate test accuracy and classification maps using the "best" model architecture;

CNN Architecture for HSI Classification



CNN architecture for HSI classification (from Paoletti et al. 2018)

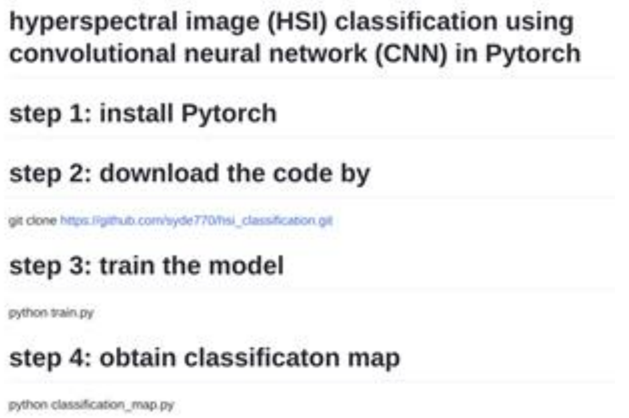
CNN Code is on Github



Use convolutional neural network (CNN) for hyperspectral image (HSI) classification

l44xu new file: LICENSE.md — Latest commit d61ed28 6 hours ago

__pycache__	add files	7 hours ago
data	deleted: data/train_0.05_val_0.2/test_mask.mat	7 hours ago
model	deleted: data/train_0.05_val_0.2/test_mask.mat	7 hours ago
LICENSE.md	new file: LICENSE.md	6 hours ago
README.md	modified: README.md	7 hours ago
classification_map.py	add files	7 hours ago
tools.py	add files	7 hours ago
train.py	add files	7 hours ago



hyperspectral image (HSI) classification using convolutional neural network (CNN) in Pytorch

step 1: install Pytorch

step 2: download the code by

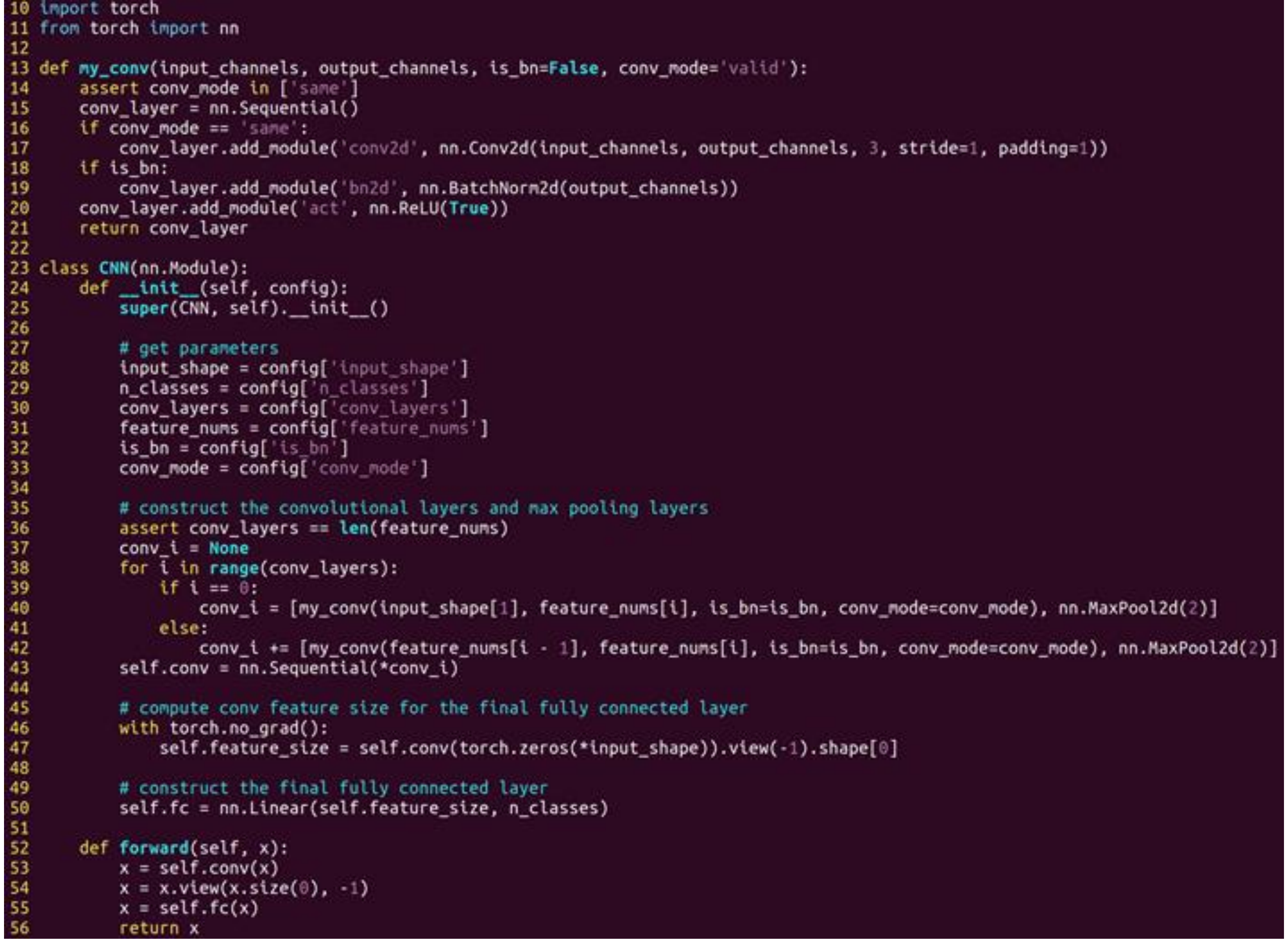
```
git clone https://github.com/syde770/hsi_classification.git
```

step 3: train the model

```
python train.py
```

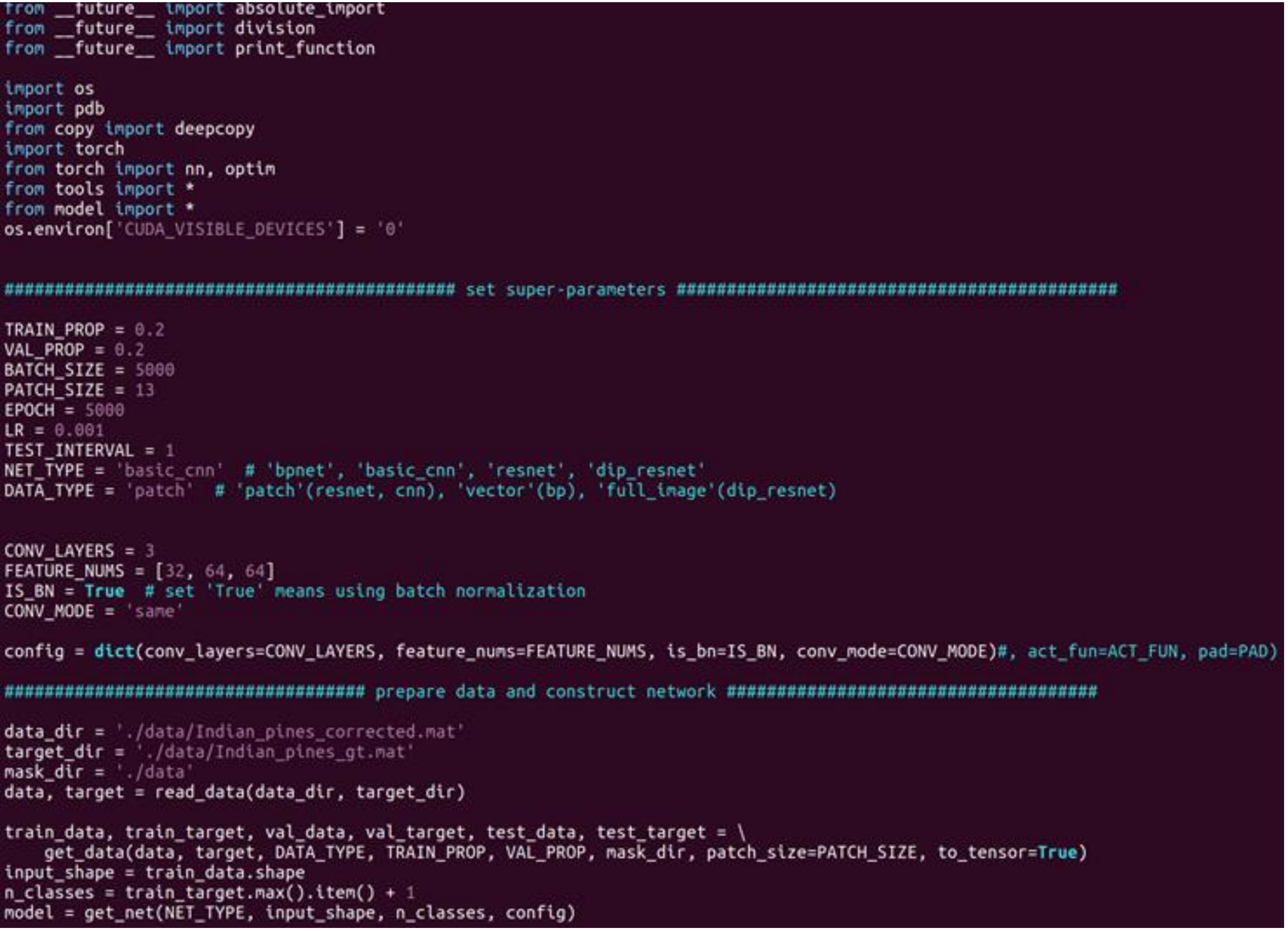
step 4: obtain classificaton map

```
python classification_map.py
```



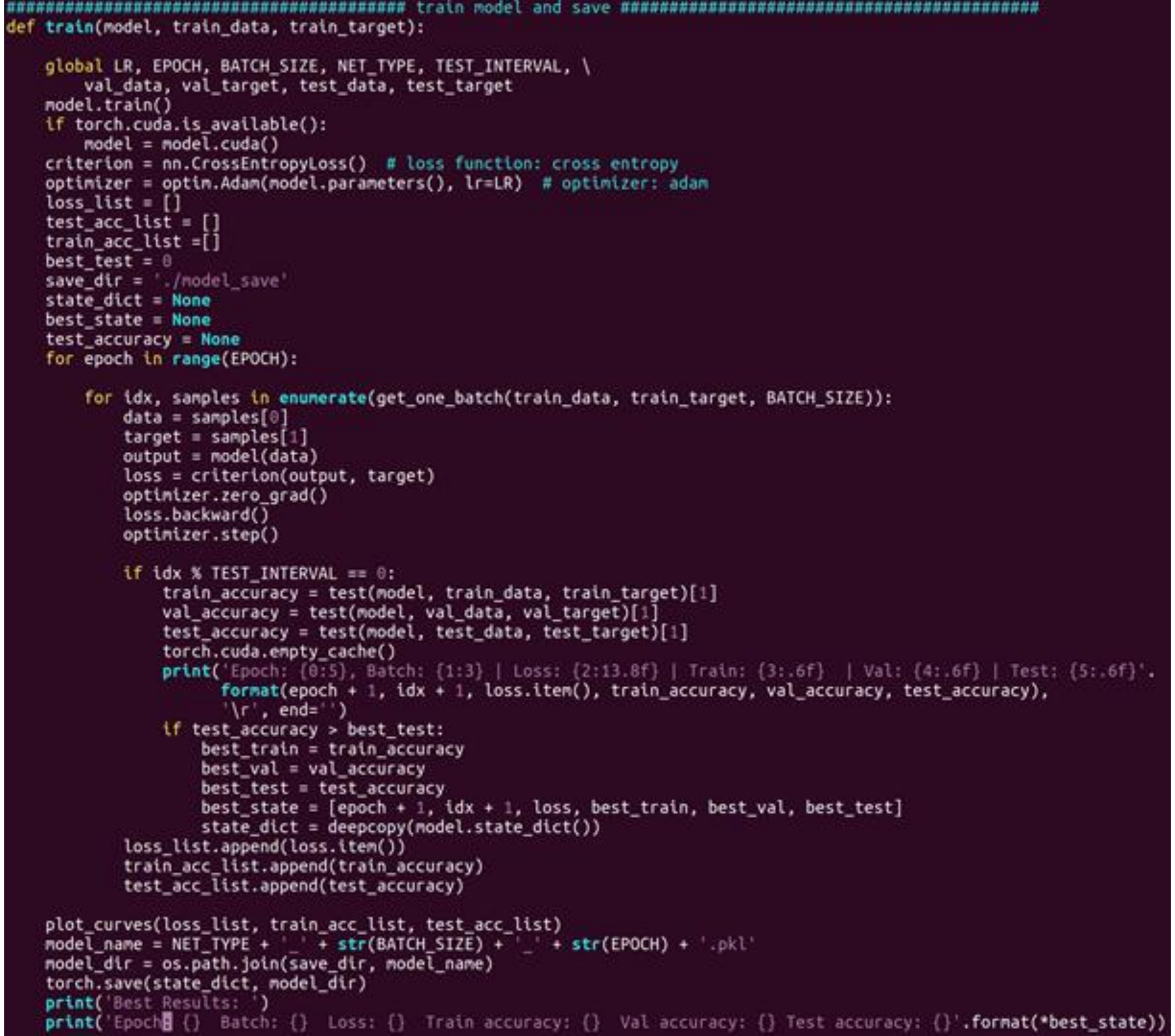
```python
import torch
from torch import nn

def my_conv(input_channels, output_channels, is_bn=False, conv_mode='valid'):
    assert conv_mode in ['same']
    conv_layer = nn.Sequential()
    if conv_mode == 'same':
        conv_layer.add_module('conv2d', nn.Conv2d(input_channels, output_channels, 3, stride=1, padding=1))
    if is_bn:
        conv_layer.add_module('bn2d', nn.BatchNorm2d(output_channels))
    conv_layer.add_module('act', nn.ReLU(True))
    return conv_layer

class CNN(nn.Module):
    def __init__(self, config):
        super(CNN, self).__init__()

        # get parameters
        input_shape = config['input_shape']
        n_classes = config['n_classes']
        conv_layers = config['conv_layers']
        feature_nums = config['feature_nums']
        is_bn = config['is_bn']
        conv_mode = config['conv_mode']

        # construct the convolutional layers and max pooling layers
        assert conv_layers == len(feature_nums)
        conv_i = None
        for i in range(conv_layers):
            if i == 0:
                conv_i = [my_conv(input_shape[1], feature_nums[i], is_bn=is_bn, conv_mode=conv_mode), nn.MaxPool2d(2)]
            else:
                conv_i += [my_conv(feature_nums[i - 1], feature_nums[i], is_bn=is_bn, conv_mode=conv_mode), nn.MaxPool2d(2)]
        self.conv = nn.Sequential(*conv_i)

        # compute conv feature size for the final fully connected layer
        with torch.no_grad():
            self.feature_size = self.conv(torch.zeros(*input_shape)).view(-1).shape[0]

        # construct the final fully connected layer
        self.fc = nn.Linear(self.feature_size, n_classes)

    def forward(self, x):
        x = self.conv(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

basic_cnn.py **defines the CNN architecture for HSI classification**



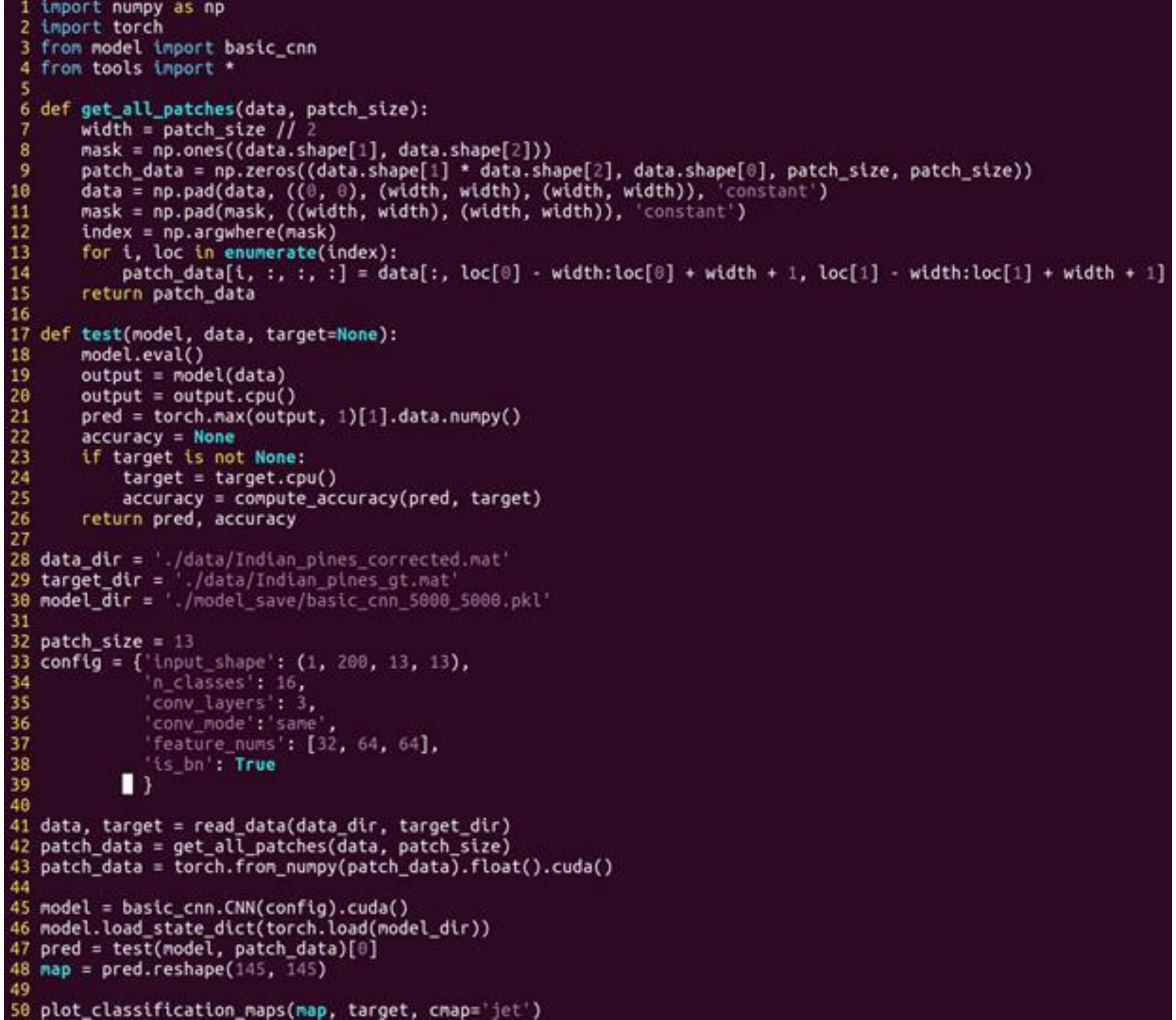
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import pdb
from copy import deepcopy
import torch
from torch import nn, optim
from tools import *
from model import *
os.environ['CUDA_VISIBLE_DEVICES'] = '0'


############################################# set super-parameters ############################################

TRAIN_PROP = 0.2
VAL_PROP = 0.2
BATCH_SIZE = 5000
PATCH_SIZE = 13
EPOCH = 5000
LR = 0.001
TEST_INTERVAL = 1
NET_TYPE = 'basic_cnn'  # 'bpnet', 'basic_cnn', 'resnet', 'dip_resnet'
DATA_TYPE = 'patch'  # 'patch'(resnet, cnn), 'vector'(bp), 'full_image'(dip_resnet)


CONV_LAYERS = 3
FEATURE_NUMS = [32, 64, 64]
IS_BN = True  # set 'True' means using batch normalization
CONV_MODE = 'same'

config = dict(conv_layers=CONV_LAYERS, feature_nums=FEATURE_NUMS, is_bn=IS_BN, conv_mode=CONV_MODE)#, act_fun=ACT_FUN, pad=PAD)

#################################### prepare data and construct network #####################################

data_dir = './data/Indian_pines_corrected.mat'
target_dir = './data/Indian_pines_gt.mat'
mask_dir = './data'
data, target = read_data(data_dir, target_dir)

train_data, train_target, val_data, val_target, test_data, test_target = \
    get_data(data, target, DATA_TYPE, TRAIN_PROP, VAL_PROP, mask_dir, patch_size=PATCH_SIZE, to_tensor=True)
input_shape = train_data.shape
n_classes = train_target.max().item() + 1
model = get_net(NET_TYPE, input_shape, n_classes, config)
```

train.py sets parameters and trains the CNN model



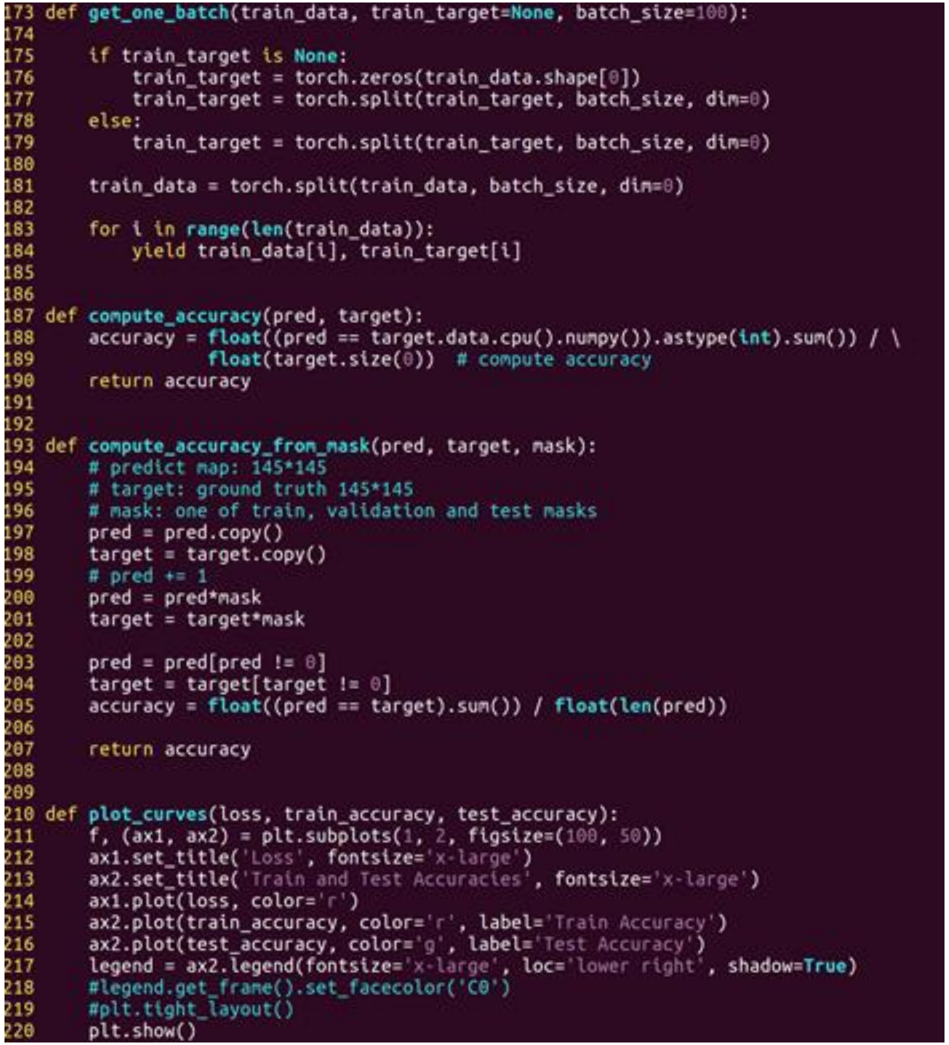
```
############################################ train model and save ###################################################
def train(model, train_data, train_target):

    global LR, EPOCH, BATCH_SIZE, NET_TYPE, TEST_INTERVAL, \
        val_data, val_target, test_data, test_target
    model.train()
    if torch.cuda.is_available():
        model = model.cuda()
    criterion = nn.CrossEntropyLoss()  # loss function: cross entropy
    optimizer = optim.Adam(model.parameters(), lr=LR)  # optimizer: adam
    loss_list = []
    test_acc_list = []
    train_acc_list =[]
    best_test = 0
    save_dir = './model_save'
    state_dict = None
    best_state = None
    test_accuracy = None
    for epoch in range(EPOCH):

        for idx, samples in enumerate(get_one_batch(train_data, train_target, BATCH_SIZE)):
            data = samples[0]
            target = samples[1]
            output = model(data)
            loss = criterion(output, target)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if idx % TEST_INTERVAL == 0:
                train_accuracy = test(model, train_data, train_target)[1]
                val_accuracy = test(model, val_data, val_target)[1]
                test_accuracy = test(model, test_data, test_target)[1]
                torch.cuda.empty_cache()
                print('Epoch: {0:5}, Batch: {1:3} | Loss: {2:13.8f} | Train: {3:.6f}  | Val: {4:.6f} | Test: {5:.6f}'.
                      format(epoch + 1, idx + 1, loss.item(), train_accuracy, val_accuracy, test_accuracy),
                      '\r', end='')
                if test_accuracy > best_test:
                    best_train = train_accuracy
                    best_val = val_accuracy
                    best_test = test_accuracy
                    best_state = [epoch + 1, idx + 1, loss, best_train, best_val, best_test]
                    state_dict = deepcopy(model.state_dict())
            loss_list.append(loss.item())
            train_acc_list.append(train_accuracy)
            test_acc_list.append(test_accuracy)

    plot_curves(loss_list, train_acc_list, test_acc_list)
    model_name = NET_TYPE + '_' + str(BATCH_SIZE) + '_' + str(EPOCH) + '.pkl'
    model_dir = os.path.join(save_dir, model_name)
    torch.save(state_dict, model_dir)
    print('Best Results: ')
    print('Epoch: {}  Batch: {}  Loss: {}  Train accuracy: {}  Val accuracy: {} Test accuracy: {}'.format(*best_state))
```

train.py sets parameters and trains the CNN model



```python
import numpy as np
import torch
from model import basic_cnn
from tools import *

def get_all_patches(data, patch_size):
    width = patch_size // 2
    mask = np.ones((data.shape[1], data.shape[2]))
    patch_data = np.zeros((data.shape[1] * data.shape[2], data.shape[0], patch_size, patch_size))
    data = np.pad(data, ((0, 0), (width, width), (width, width)), 'constant')
    mask = np.pad(mask, ((width, width), (width, width)), 'constant')
    index = np.argwhere(mask)
    for i, loc in enumerate(index):
        patch_data[i, :, :, :] = data[:, loc[0] - width:loc[0] + width + 1, loc[1] - width:loc[1] + width + 1]
    return patch_data

def test(model, data, target=None):
    model.eval()
    output = model(data)
    output = output.cpu()
    pred = torch.max(output, 1)[1].data.numpy()
    accuracy = None
    if target is not None:
        target = target.cpu()
        accuracy = compute_accuracy(pred, target)
    return pred, accuracy

data_dir = './data/Indian_pines_corrected.mat'
target_dir = './data/Indian_pines_gt.mat'
model_dir = './model_save/basic_cnn_5000_5000.pkl'

patch_size = 13
config = {'input_shape': (1, 200, 13, 13),
          'n_classes': 16,
          'conv_layers': 3,
          'conv_mode':'same',
          'feature_nums': [32, 64, 64],
          'is_bn': True
        }

data, target = read_data(data_dir, target_dir)
patch_data = get_all_patches(data, patch_size)
patch_data = torch.from_numpy(patch_data).float().cuda()

model = basic_cnn.CNN(config).cuda()
model.load_state_dict(torch.load(model_dir))
pred = test(model, patch_data)[0]
map = pred.reshape(145, 145)

plot_classification_maps(map, target, cmap='jet')
```

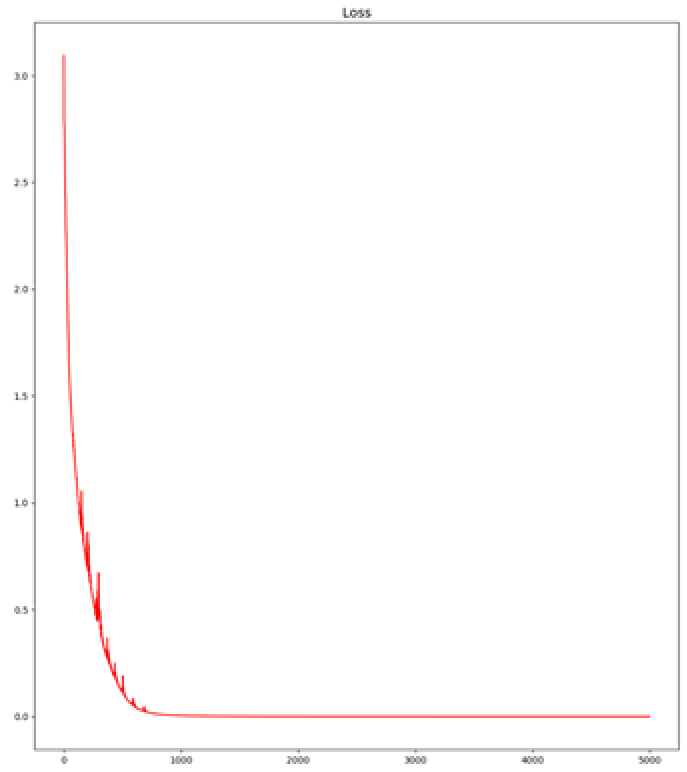
classification_map.py predicts all pixels on HSI and generates classification map using the trained model by train.py

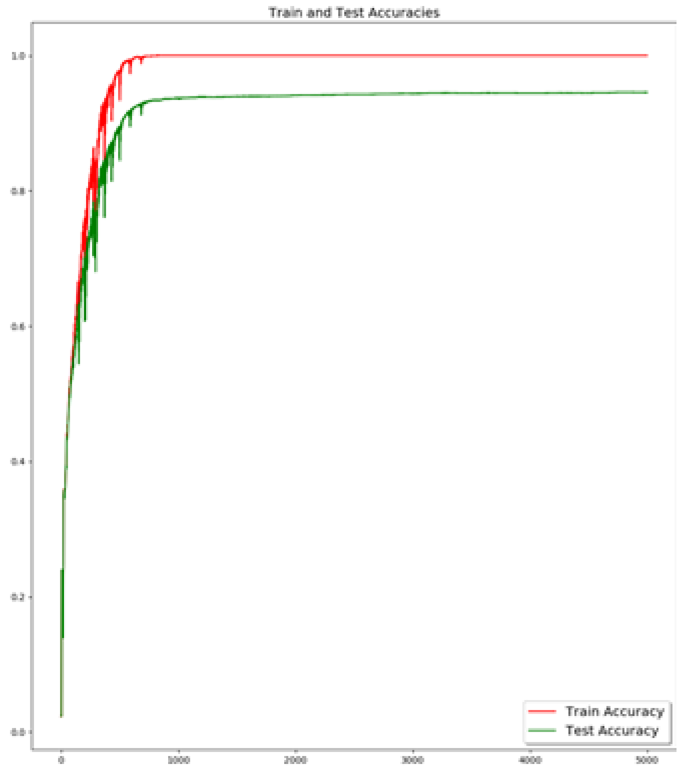


```python
def get_one_batch(train_data, train_target=None, batch_size=100):

    if train_target is None:
        train_target = torch.zeros(train_data.shape[0])
        train_target = torch.split(train_target, batch_size, dim=0)
    else:
        train_target = torch.split(train_target, batch_size, dim=0)

    train_data = torch.split(train_data, batch_size, dim=0)

    for i in range(len(train_data)):
        yield train_data[i], train_target[i]


def compute_accuracy(pred, target):
    accuracy = float((pred == target.data.cpu().numpy()).astype(int).sum()) / \
               float(target.size(0))  # compute accuracy
    return accuracy


def compute_accuracy_from_mask(pred, target, mask):
    # predict map: 145*145
    # target: ground truth 145*145
    # mask: one of train, validation and test masks
    pred = pred.copy()
    target = target.copy()
    # pred += 1
    pred = pred*mask
    target = target*mask

    pred = pred[pred != 0]
    target = target[target != 0]
    accuracy = float((pred == target).sum()) / float(len(pred))

    return accuracy


def plot_curves(loss, train_accuracy, test_accuracy):
    f, (ax1, ax2) = plt.subplots(1, 2, figsize=(100, 50))
    ax1.set_title('Loss', fontsize='x-large')
    ax2.set_title('Train and Test Accuracies', fontsize='x-large')
    ax1.plot(loss, color='r')
    ax2.plot(train_accuracy, color='r', label='Train Accuracy')
    ax2.plot(test_accuracy, color='g', label='Test Accuracy')
    legend = ax2.legend(fontsize='x-large', loc='lower right', shadow=True)
    #legend.get_frame().set_facecolor('C0')
    #plt.tight_layout()
    plt.show()
```

tools.py implements some functions for generating training samples and visualization.

Loss, Train Acc and Test Acc over Iterations



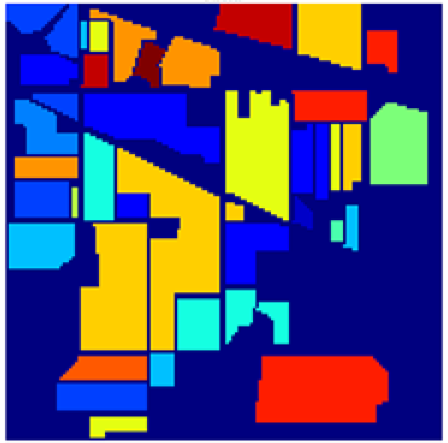


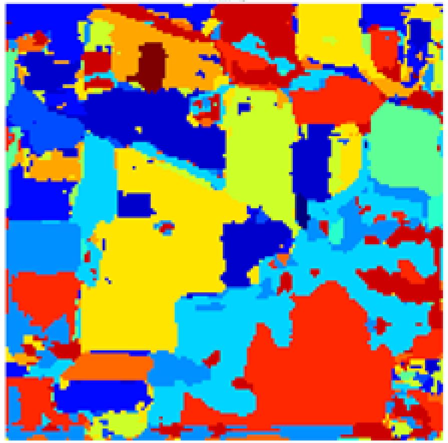
(1) Loss keeps decreasing over stochastic gradient descent (SGD) iterations;

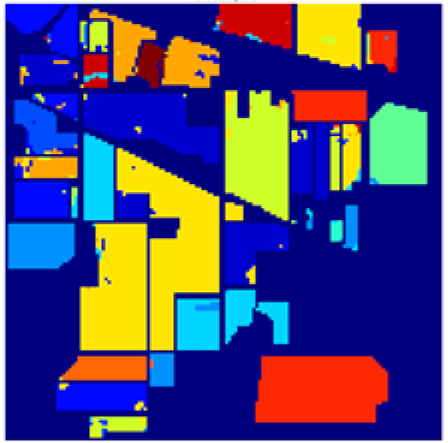
(2) Both train and test accuracies keep increasing;

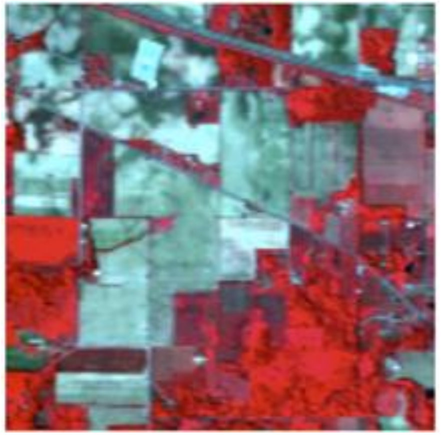
(3) Training accuracy is higher than test accuracy over iterations;

Classification Results via CNN (with 20% pixels used as training samples)


Ground Truth


CNN Full Map (OA:94.5%)


CNN No Background (OA: 94.5%)


False Color Image

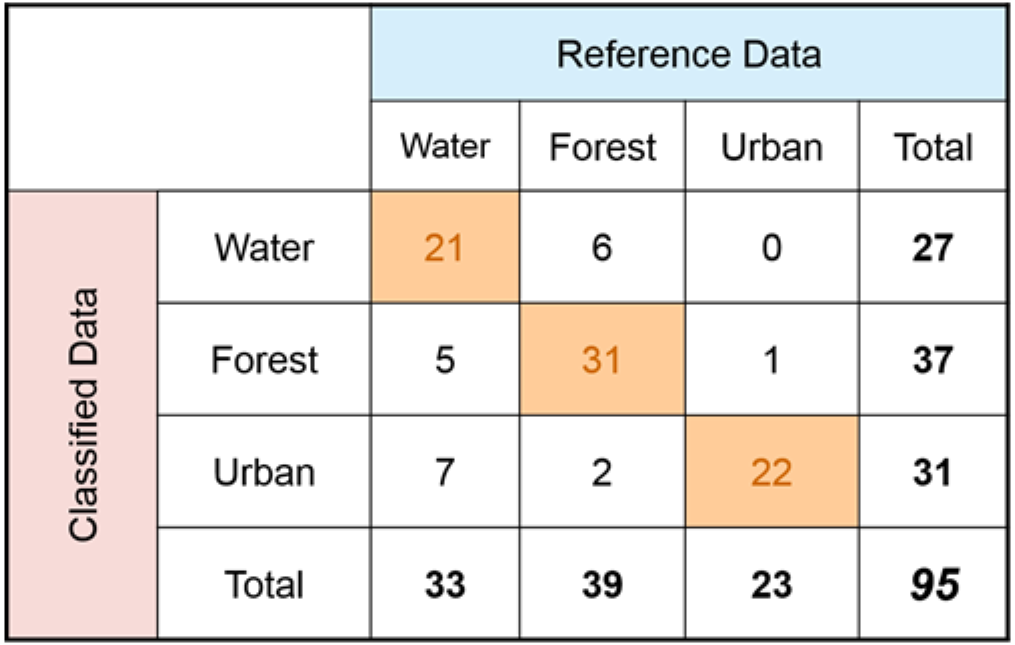
🟦	1	Alfalfa	🟩	9	Oats
🟦	2	Corn-notill	🟨	10	Soybean-notill
🟦	3	Corn-mintill	🟨	11	Soybean-mintill
🟦	4	Corn	🟧	12	Soybean-clean
🟦	5	Grass-pasture	🟧	13	Wheat
🟦	6	Grass-trees	🟥	14	Woods
🟩	7	Grass-pasture-mowed	🟥	15	Buildings-Grass-Trees-Drives
🟩	8	Hay-windrowed	🟫	16	Stone-Steel-Towers

(1) Overall, the classification maps generated by CNN match the ground truth map very well;

(2) CNN full map generally delineates quite well the edges in the false color image, although there is still room for improvement in terms of edge preservation;

(3) Overall accuracy of 94.5% is very high;

Quantitative Classification Performance Evaluation



		Reference Data			
		Water	Forest	Urban	Total
Classified Data	Water	21	6	0	**27**
	Forest	5	31	1	**37**
	Urban	7	2	22	**31**
	Total	**33**	**39**	**23**	***95***

1. Overall accuracy (OA)

OA = (21 + 31+ 22)/**95** = 77.9%

2. Producer's accuracy (PA)

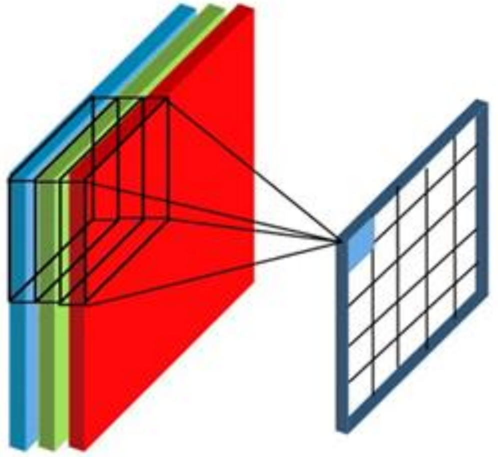
PA_{water}= 21/**33** = 64%

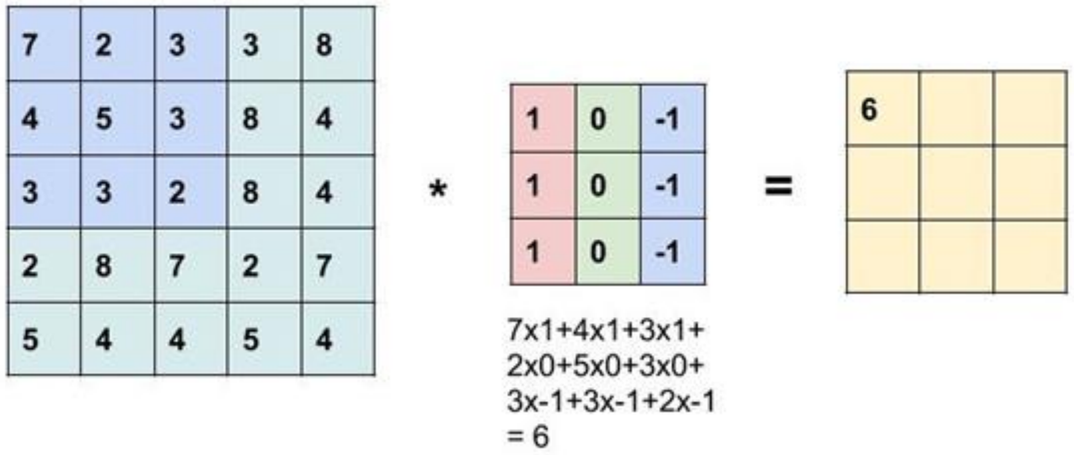
3. User's accuracy (UA)

UA_{water}= 21/**27**= 78%

How to calculate PA and UA for Forest and Urban?

Convolutional neural network (CNN)

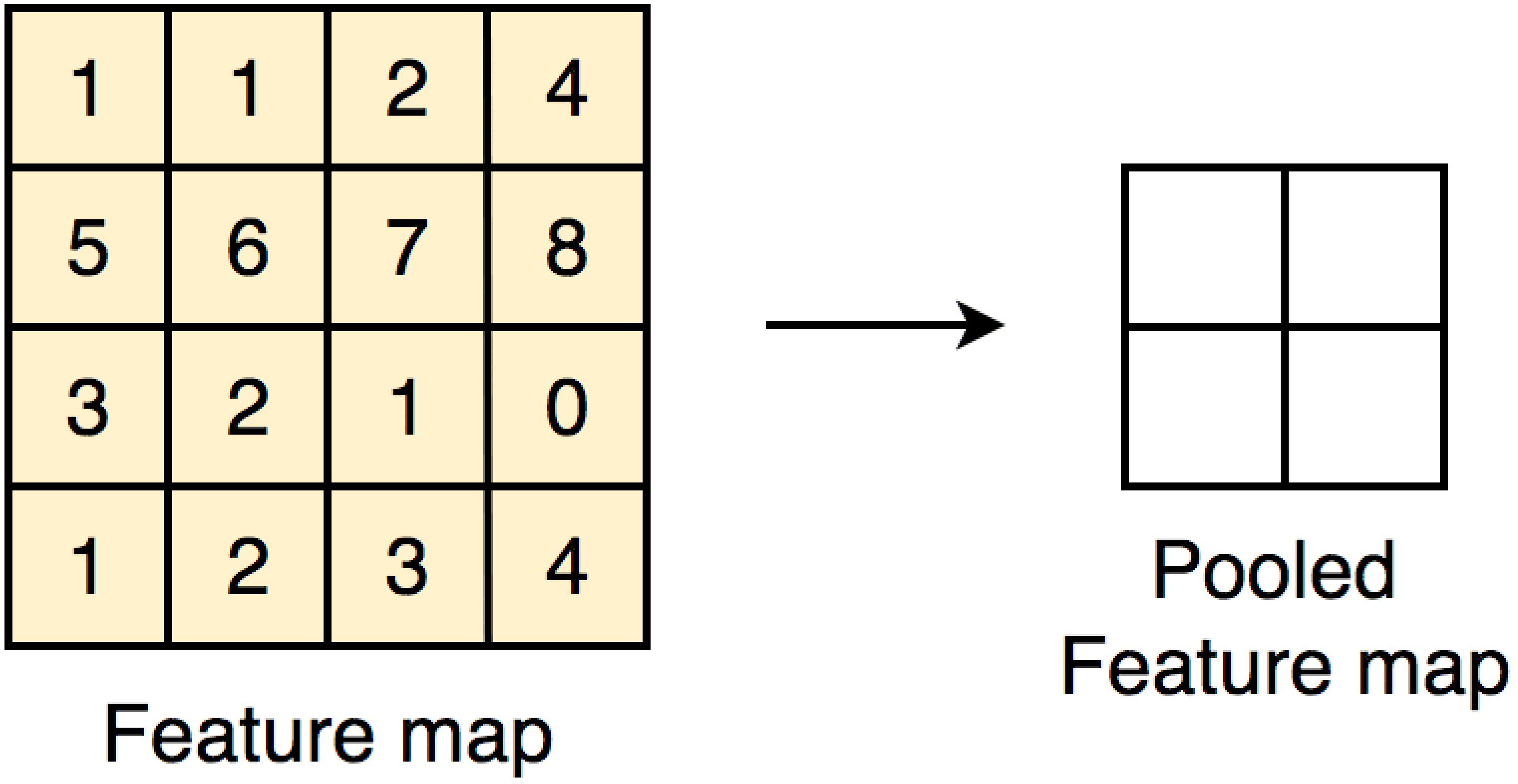




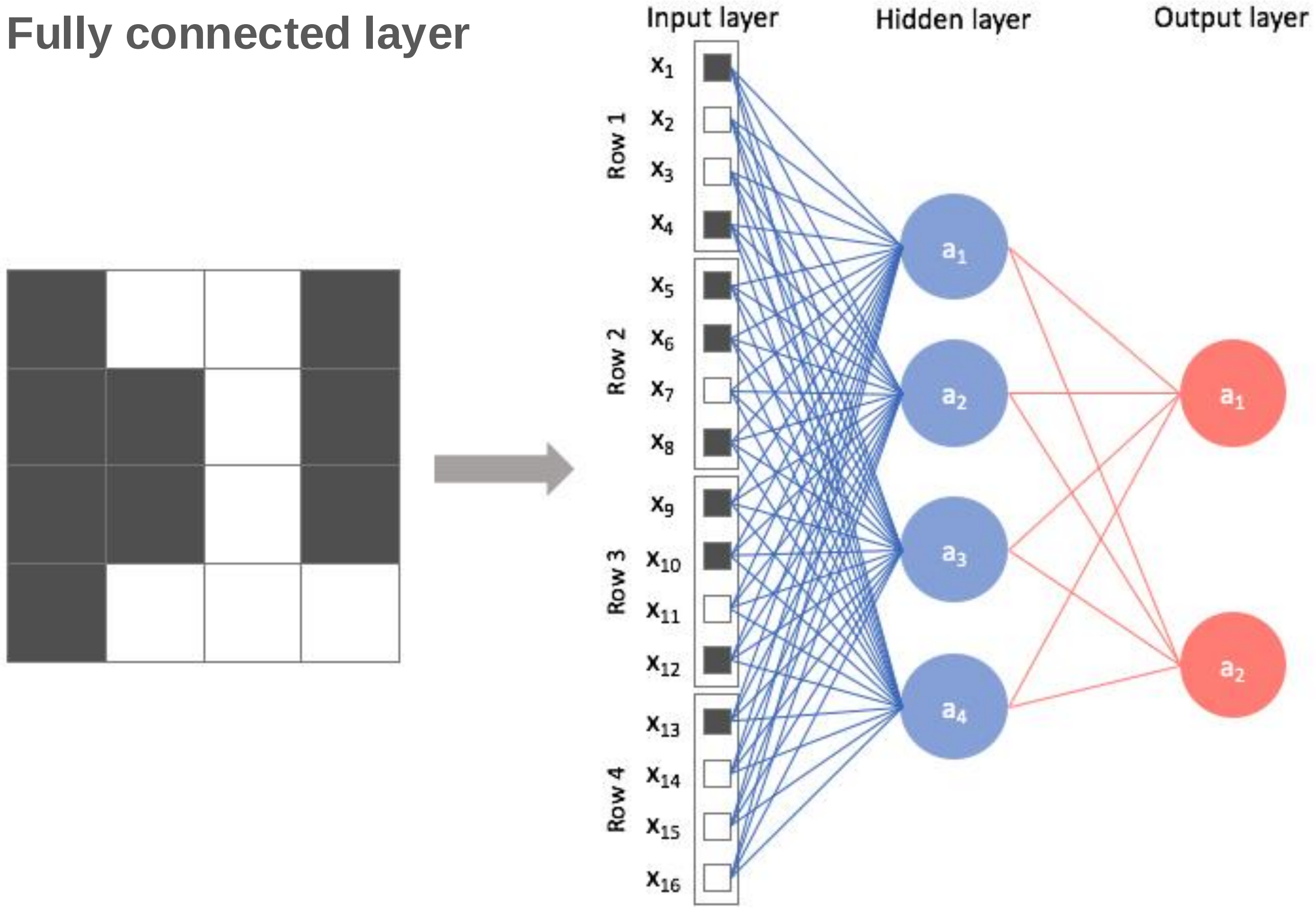
$$\begin{bmatrix} 7 & 2 & 3 & 3 & 8 \\ 4 & 5 & 3 & 8 & 4 \\ 3 & 3 & 2 & 8 & 4 \\ 2 & 8 & 7 & 2 & 7 \\ 5 & 4 & 4 & 5 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 6 & & \\ & & \\ & & \end{bmatrix}$$

7x1+4x1+3x1+
2x0+5x0+3x0+
3x-1+3x-1+2x-1
= 6

Max pooling layer



Fully connected layer



Questions?