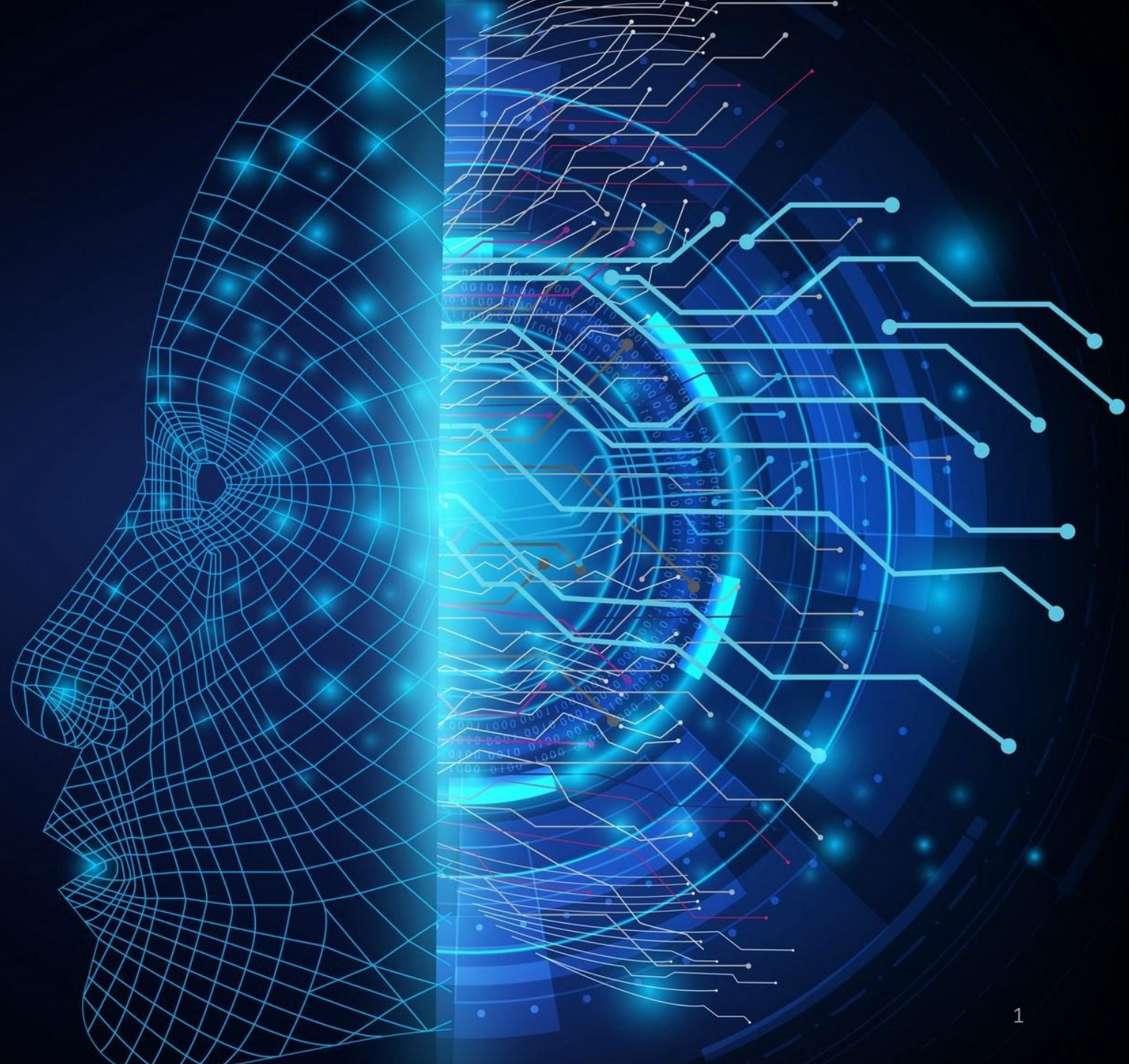
Deep Feature Extraction and Classification for Hyperspectral Imagery

Behnood Rasti

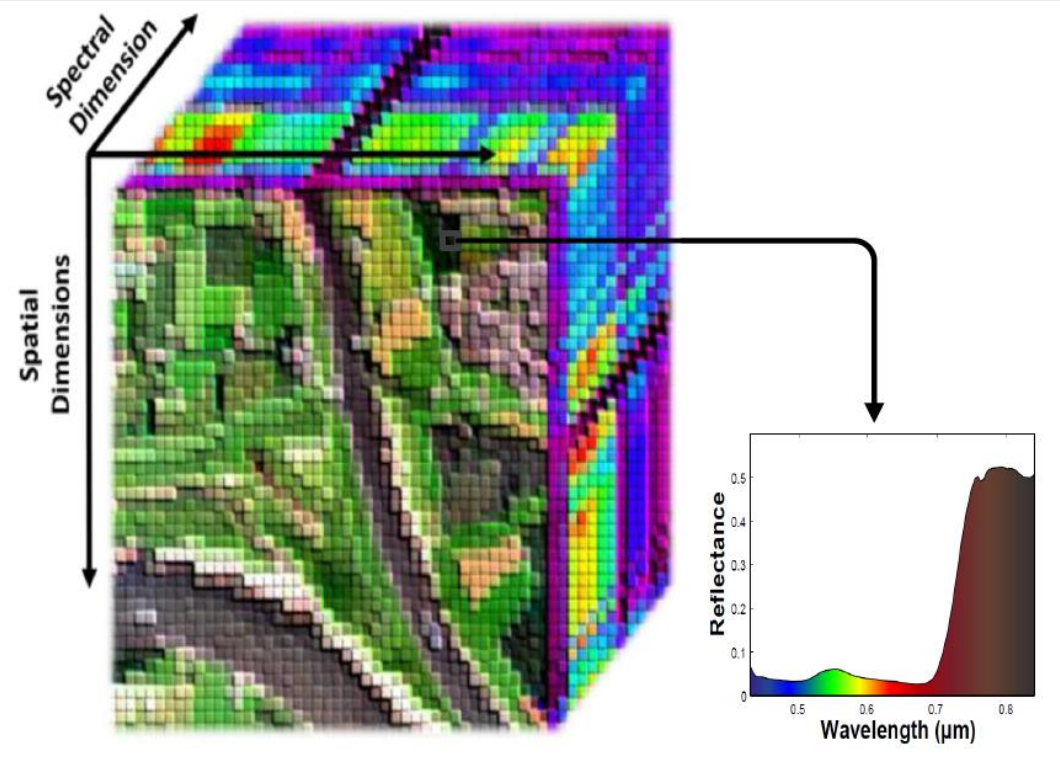
PhD., Electrical and Computer Engineering



Objectives

- Concept of Feature Extraction
- Shallow Feature Extraction
- Deep Feature Extraction
- Basic Deep Learning Architectures
- Shallow vs. Deep Feature Extraction
- Introducing HyFTech (Hyperspectral deep and shallow feature extraction toolbox)

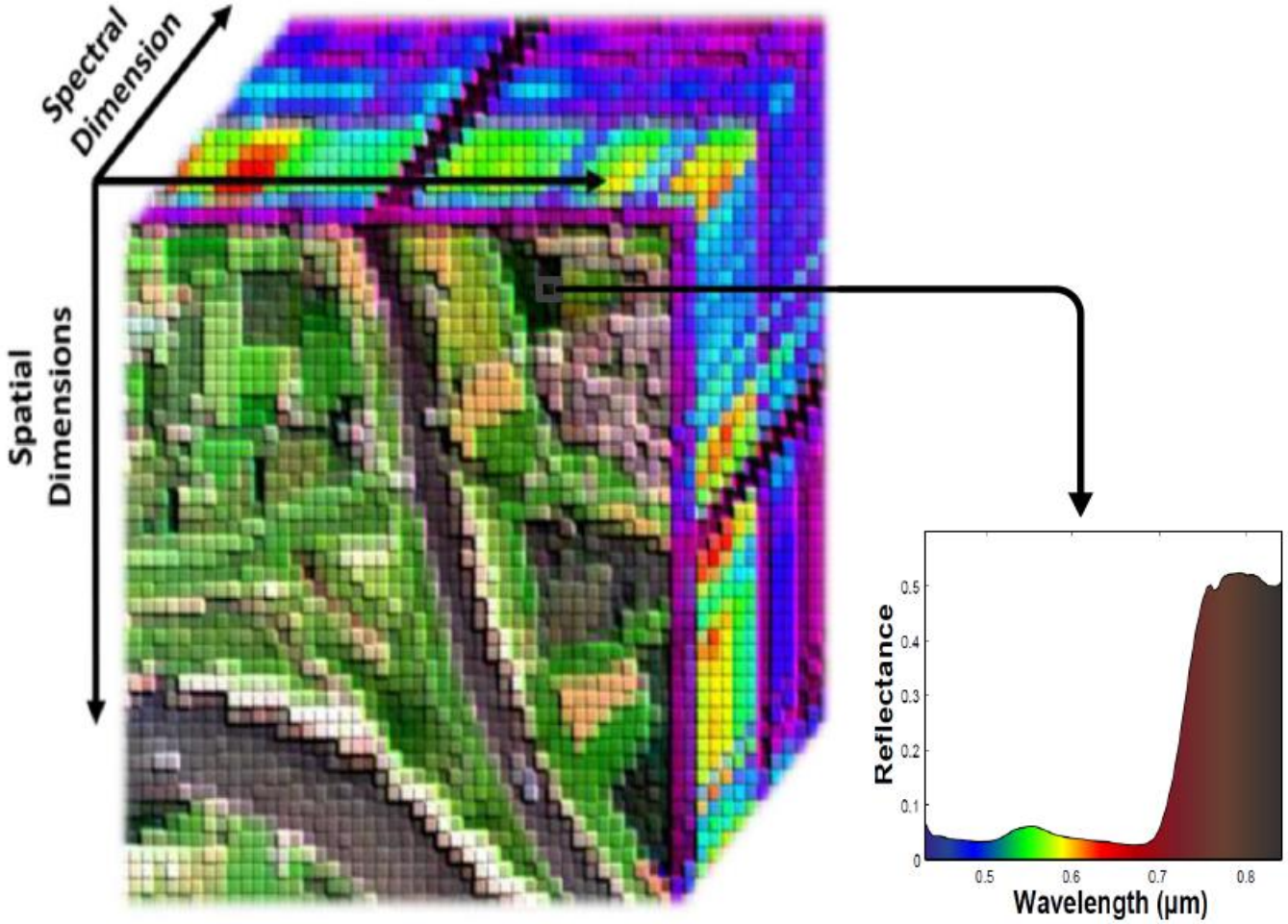
Hyperspectral Image



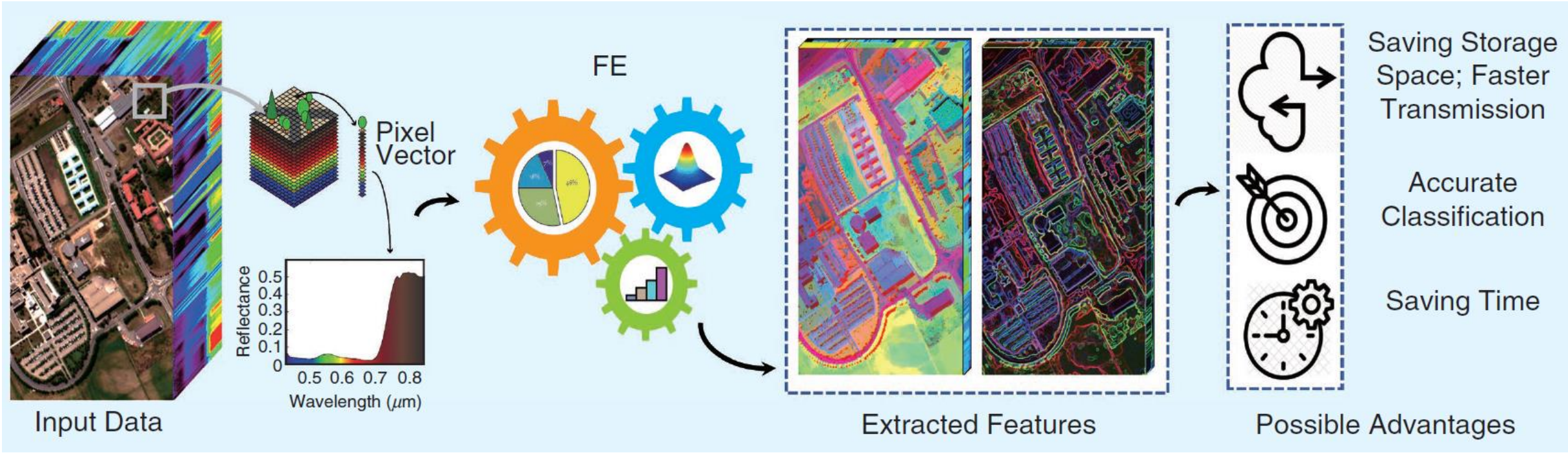
- **Hyperspectral cameras:** provide contiguous electromagnetic spectra ranging from visible over near-infrared to shortwave infrared spectral bands (from 0.3 m to 2.5 m).
- The spectral signature: allowing to distinguish between materials with different characteristics.

Hyperspectral Challenges

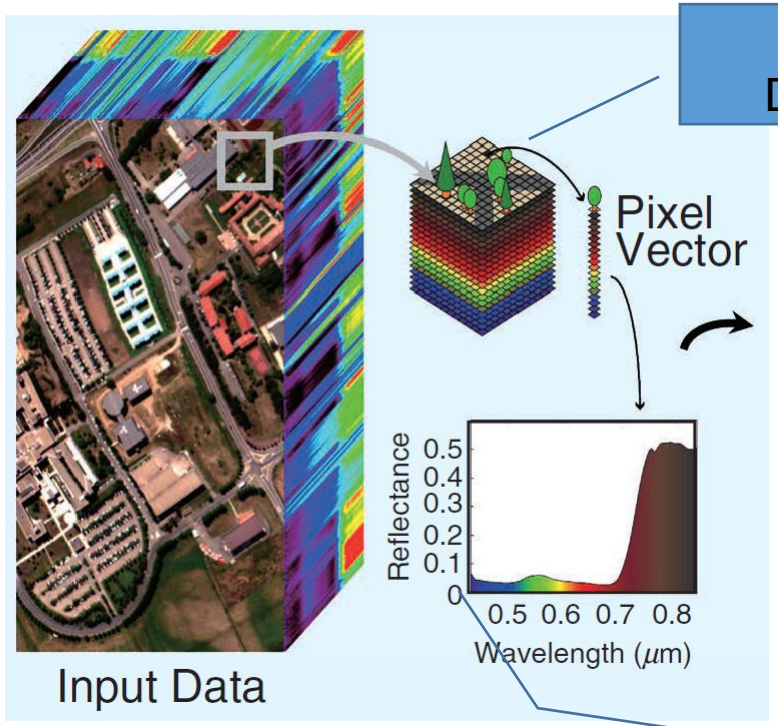
- Hyperspectral sensors have different spatial resolution, spectral resolution, and spectral ranging.
- Hyperparameter tuning and architecture design is challenging
- Limited training samples

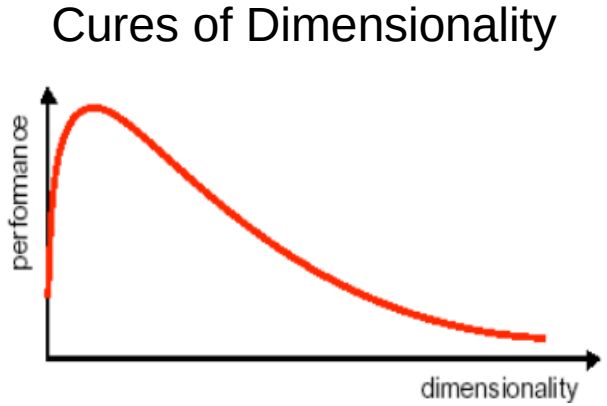


What is Feature Extraction? Why FE?



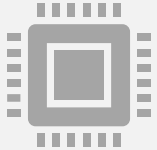
High Dimension is Challenging





Downsides of Spectral Features

High dimensionality (Curse of dimensionality)

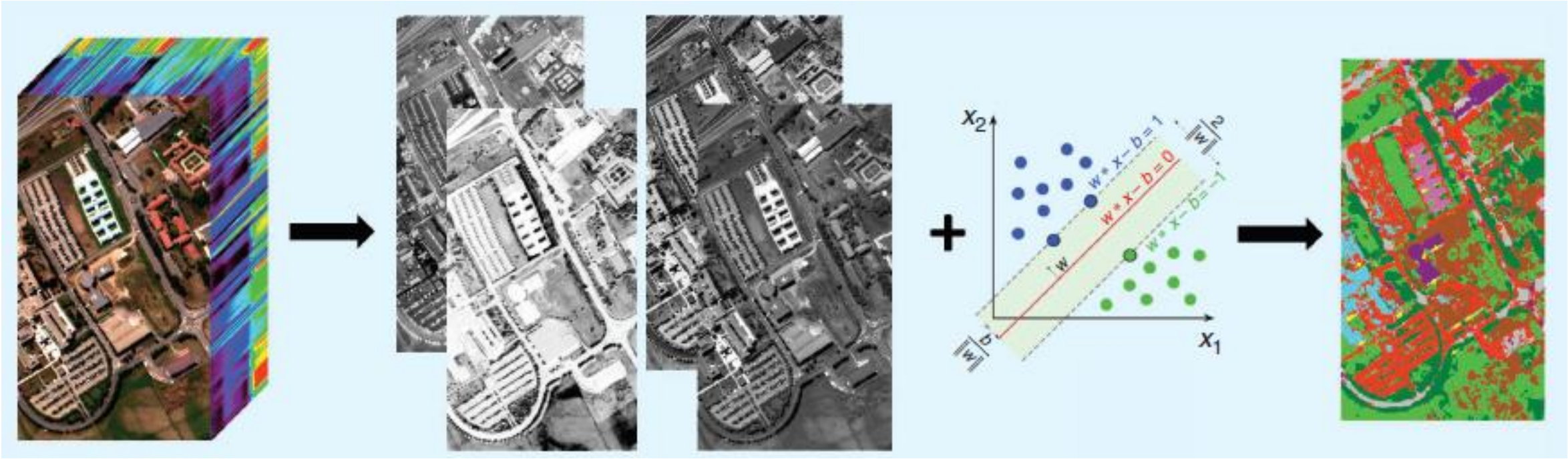


Redundancy (Saving storage and computations)

No spatial dependency (HSI contains low-spatial info.)

Conventional ML for Classification of HSI

- Basic idea: Applying the following steps sequentially:
  1. Feature Extraction/Reduction
  2. Supervised Classification (RF/SVM)



Shallow Feature Extraction

1. Unsupervised Feature Techniques (UFE):
   - Do not incorporate the ground reference/ labeled samples
   - Capturing the intrinsic characteristics of the HSI data, such as geometric, spatial, or spectral information
2. Supervised Feature Techniques (SFE):
   - Relying on the labeled samples
   - Extracting class-separable features

Norm

- A function which maps a vector space to the nonnegative real numbers
- p-norm of vector $\mathbf{x} \in \mathbb{R}^n$

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

- ℓ_1

$$\|\boldsymbol{x}\|_1 := \sum_{i=1}^{n} |x_i|$$

ℓ_2

$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$$

Unsupervised Feature Extraction

- Generally, UFE is searching for a projection matrix (subspace bases) while capturing the intrinsic characteristics of data points (Y)

$$Z = V^T Y$$

- PCA finds a subspace which maximize the variance

$$\max_{\mathbf{V}} \frac{\mathbf{V}^T \mathbf{C} \mathbf{V}}{\mathbf{V}^T \mathbf{V}}$$

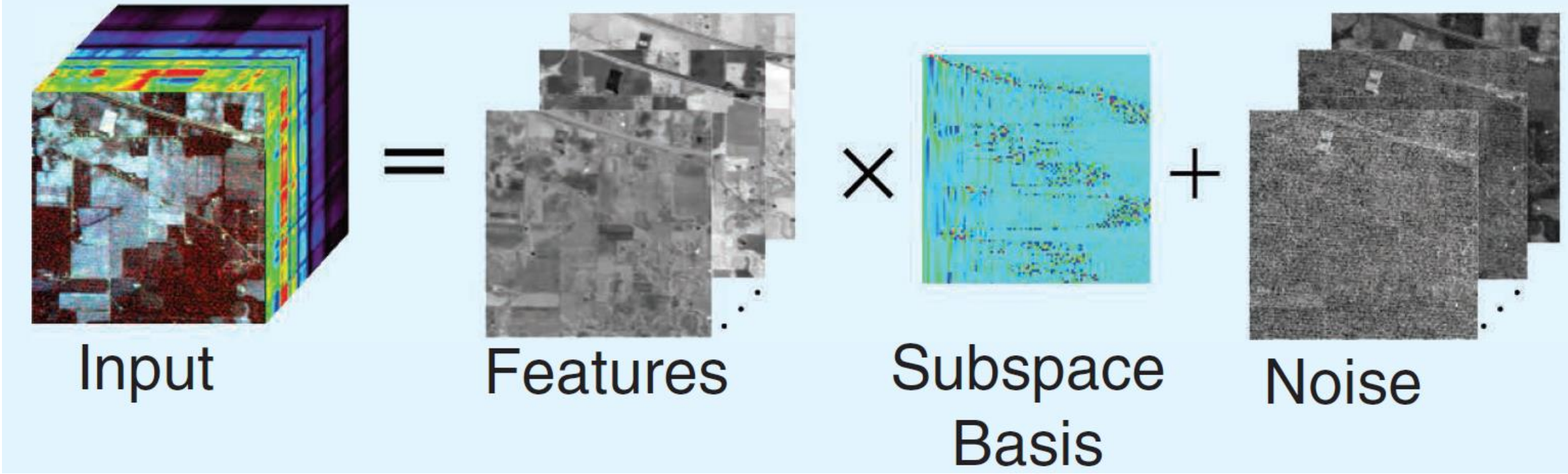
  - C is the covariance matrix
- MNF seeks a projection to maximize the SNR

$$\max_{\mathbf{V}} \frac{\mathbf{V}^T \mathbf{C} \mathbf{V}}{\mathbf{V}^T \mathbf{C}_n \mathbf{V}},$$

  - C_n is the noise covariance matrix

Low-Rank Reconstruction-Based Techniques

OTVCA, WSRRR, and SSLRA



Orthogonal Total Variation Component Analysis (OTVCA)

- HSI is modeled as

$$Y = V^T F + N$$

  - $Y \in \mathbb{R}^{p \times n}$: Observed data
  - $V \in \mathbb{R}^{r \times p}$: The bases for the subspace
  - $F \in \mathbb{R}^{r \times n}$: Features
  - N $\in \mathbb{R}^{p \times n}$: Noise and model error

- OTVCA suggests an iterative algorithm to solve the optimization

$$(\hat{V}, \hat{F}) = arg \min_{V,F} \frac{1}{2} ||Y - V^T F||_F^2 + \lambda \sum_{j=1}^{r} TV(vec^{-1}(f_{(j)}^T)) \text{ st. } VV^T = I$$

Supervised Feature Extraction

- Generally, SFE searches for an optimal projection or transformation matrix $P \in \mathbb{R}^{p \times r}$ by optimizing certain class-relevant separation criteria given a training set containing m labeled samples

$$\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$$

  - Where the samples are $\mathrm{X_m} = \{\mathrm{x_i}\}_{\mathrm{i=1}}^{m} \in \mathbb{R}^{p \times m}$
- Then, the data samples (Y) are projected into a subspace and the features ($Z \in \mathbb{R}^{r \times n}$) given by

$$Z = P^T Y$$

(Fisher's) Linear Discriminant Analysis

- Estimating the linear projection matrix P that

$$\max_P \frac{P^T S_b P}{P^T S_w P}$$

- S_b: Between-class covariance matrix

$$S_b = \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T$$

- S_w: Within-class covariance matrix

$$S_w = \sum_{k=1}^{K} \sum_{n \in C_k} (x_n - m_k)(x_n - m_k)^T$$

- m_k: mean of class k
- m: mean of training samples
- N_k: Number of samples in class k

Joint Learning

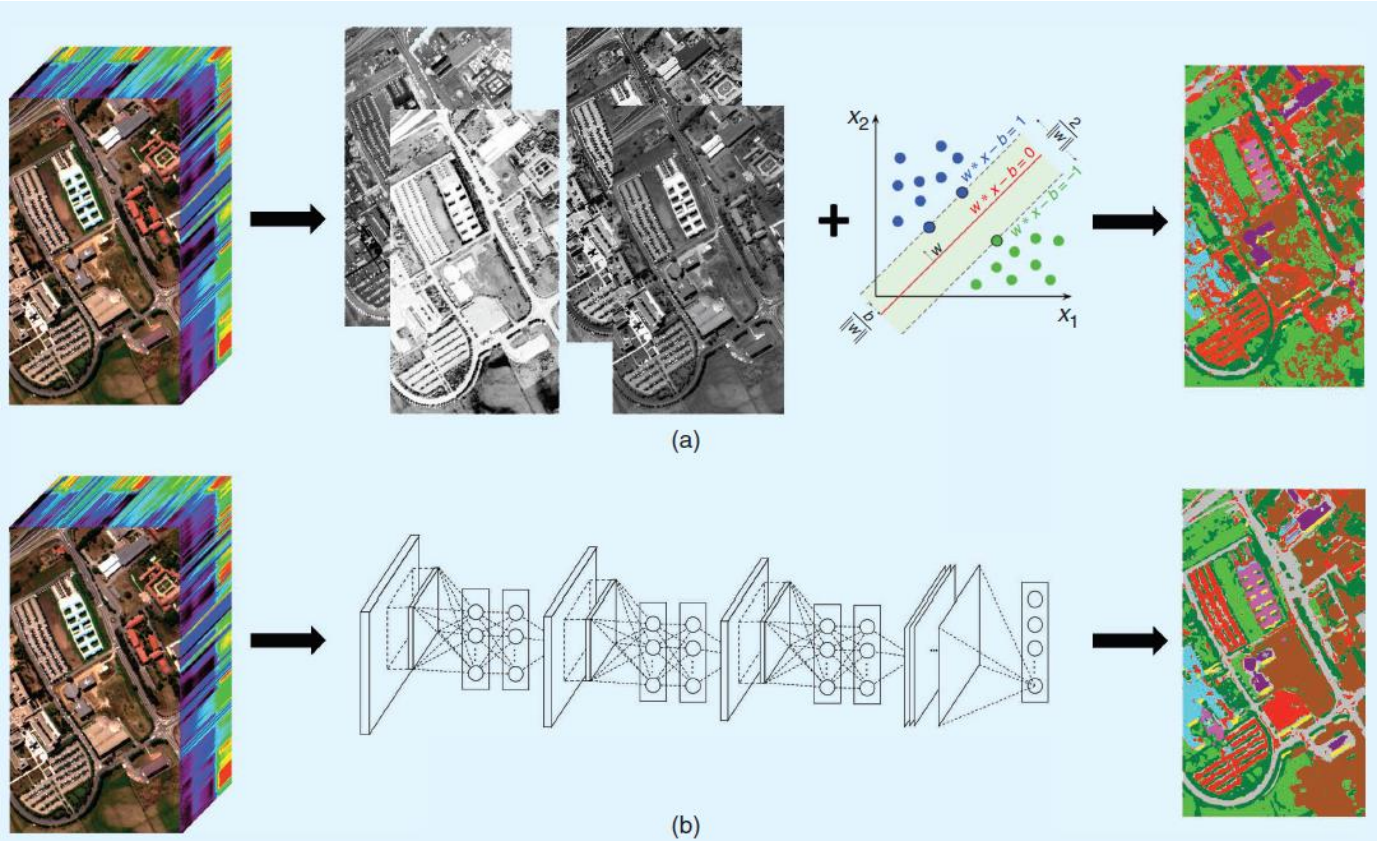
- JL: To simultaneously learn the subspace bases ($\Theta \in \mathbb{R}^{r\times p}$) and the regression matrix ($\mathrm{P_k} \in \mathbb{R}^{k\times r}$, k is the number of classes) that are applied on the samples ($\mathrm{X_m} = \{\mathrm{x_i}\}_{\mathrm{i=1}}^{m} \in \mathbb{R}^{p\times m}$) and provides the labels

$$\min_{\mathbf{P}_k,\boldsymbol{\Theta}} \| \mathbf{Y}_l - \mathbf{P}_k \boldsymbol{\Theta} \mathbf{X}_m \|_F^2 + \frac{\alpha}{2} \| \mathbf{P}_k \|_F^2 \quad \text{s.t.} \quad \boldsymbol{\Theta}\boldsymbol{\Theta}^{\mathrm{T}} = \mathbf{I},$$

- $Y_l \in \mathbb{R}^{k\times m}$: one-hot encoded label matrix

Hyperspectral Feature Extraction

Shallow vs. Deep Feature Extraction



Deep Feature Extraction

Shallow FE techniques often require careful engineering and domain knowledge of experts

Deep networks can automatically learn high-level features from raw data in a hierarchical fashion

Deep features are more discriminative, abstract, and robust than shallow ones.

Basic networks (AE,CAE,CNN, RNN, CRNN)

Challenges of Deep Feature Extraction

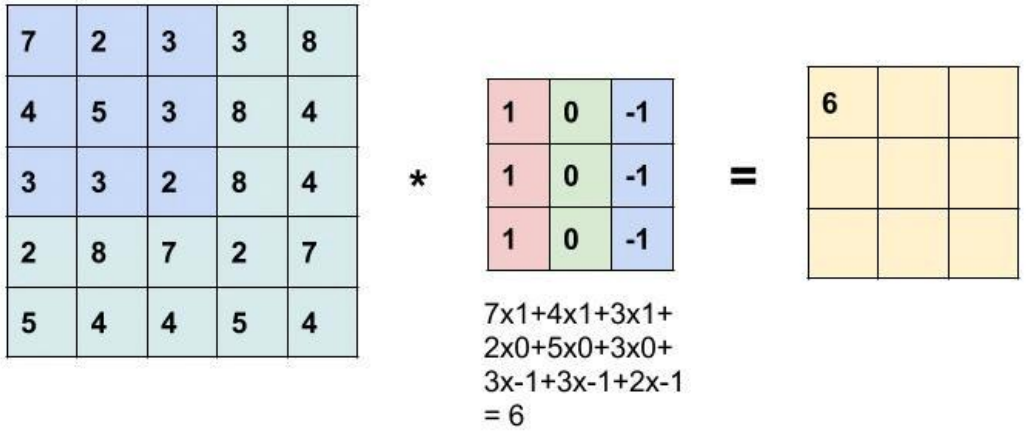
Limited number of training samples

Selection of the architecture

Hyperparameter tuning

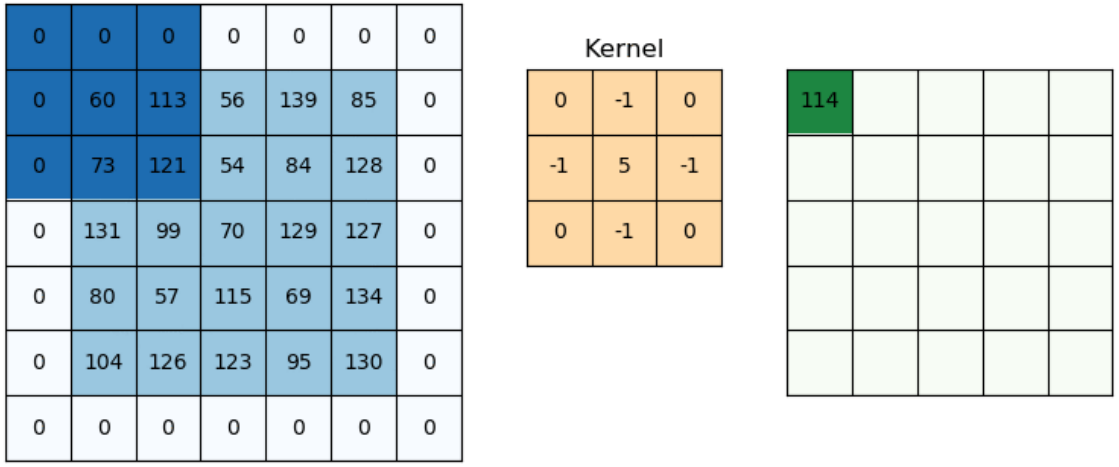
Question: Would DL-based approaches outperform shallow ones in terms of classification accuracy?

Convolution



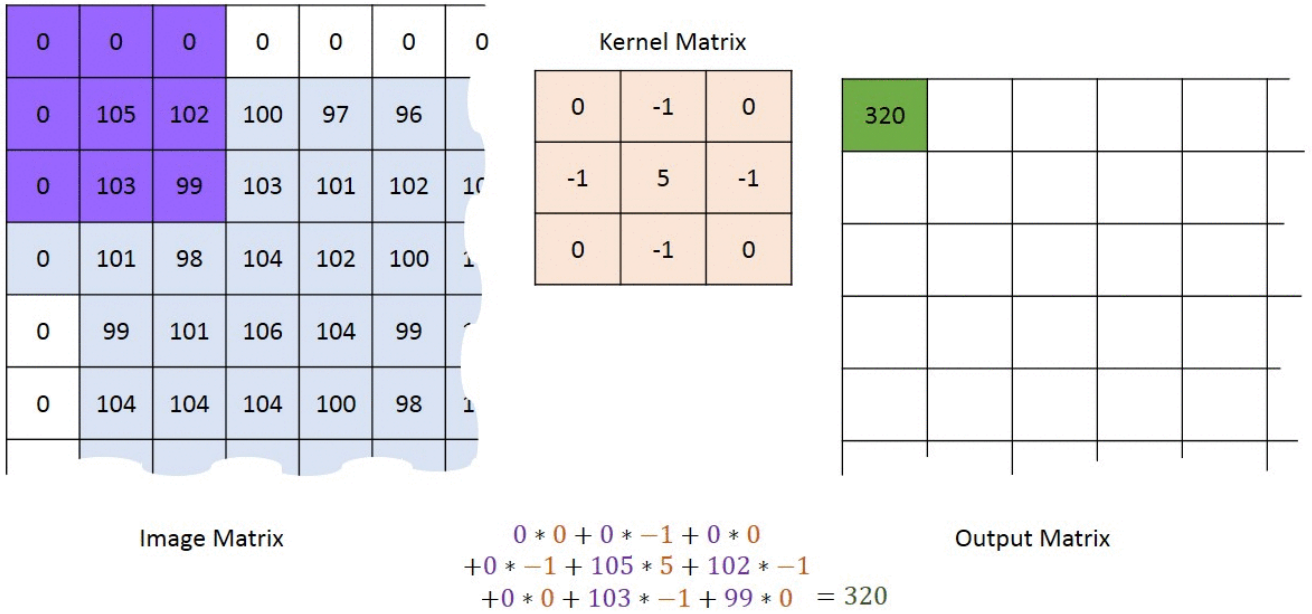
https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e

Convolution with Padding



https://learnopencv.com/deep-convolutional-gan-in-pytorch-and-tensorflow/

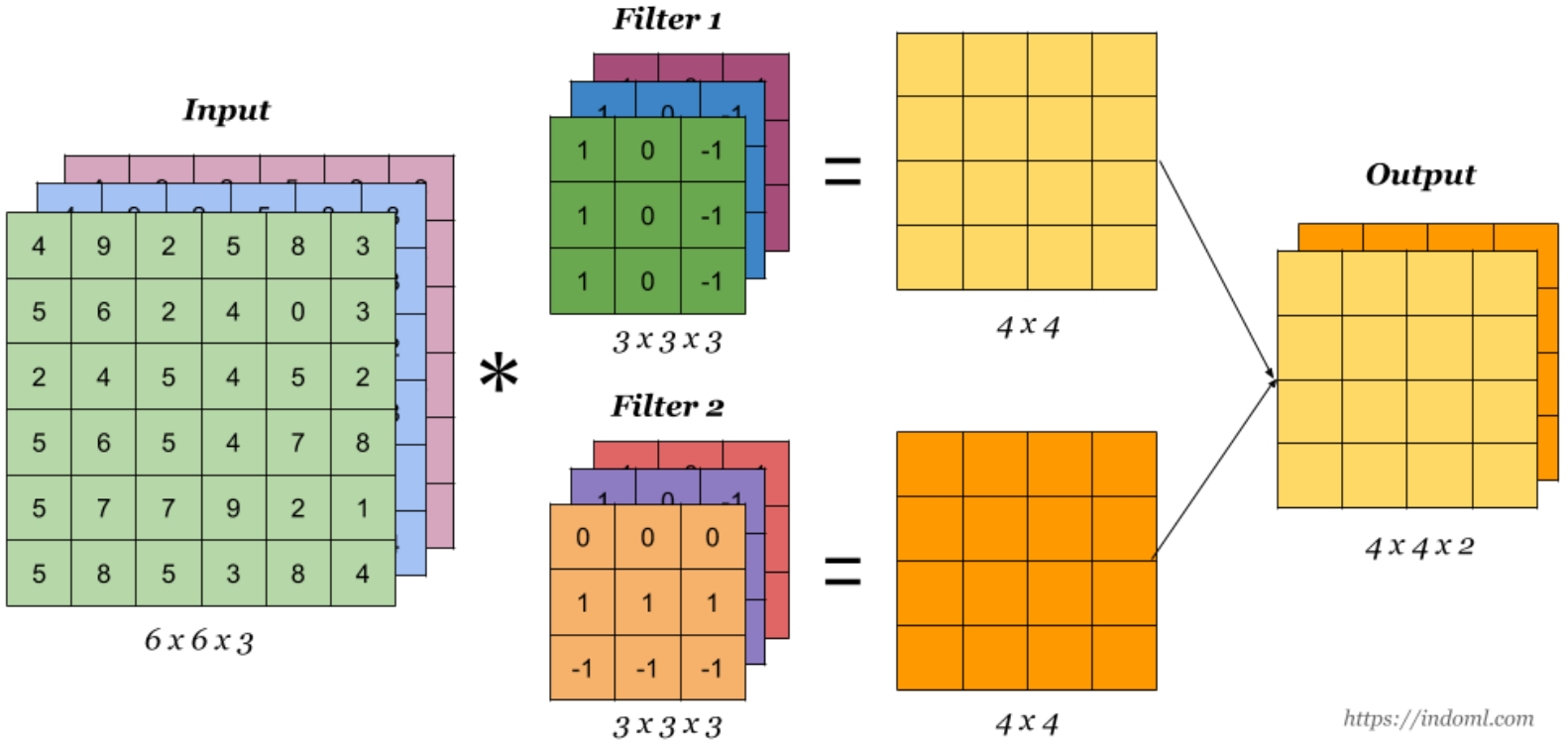
Strided Convolution



Convolution with horizontal and vertical strides = 2

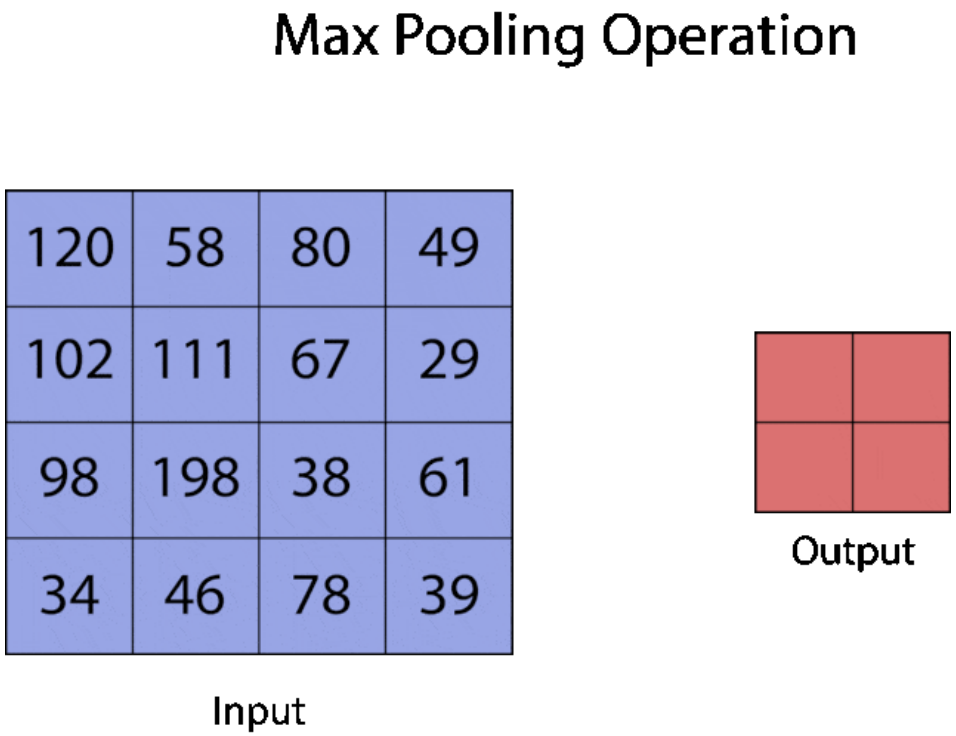
https://vernlium.github.io/2018/10/15/coursera-deeplearning-ai-c4-week1/

Convolutions Over Bands



https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

Pooling

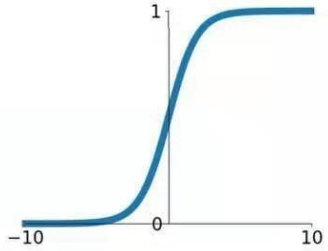


https://deeplearningdemystified.com/article/cv-1

Activation Functions

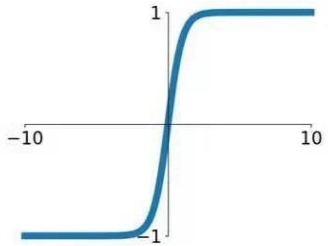
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



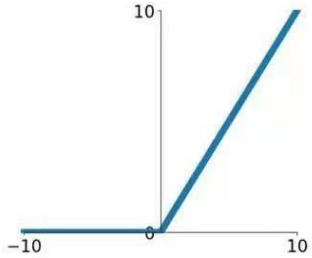
tanh

$$\tanh(x)$$



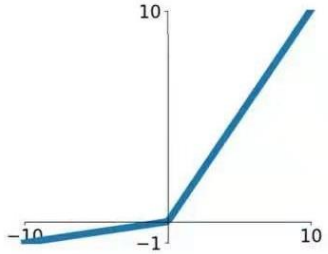
ReLU

$$\max(0, x)$$



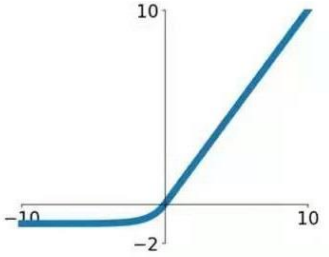
Leaky ReLU

$$\max(0.1x, x)$$



ELU

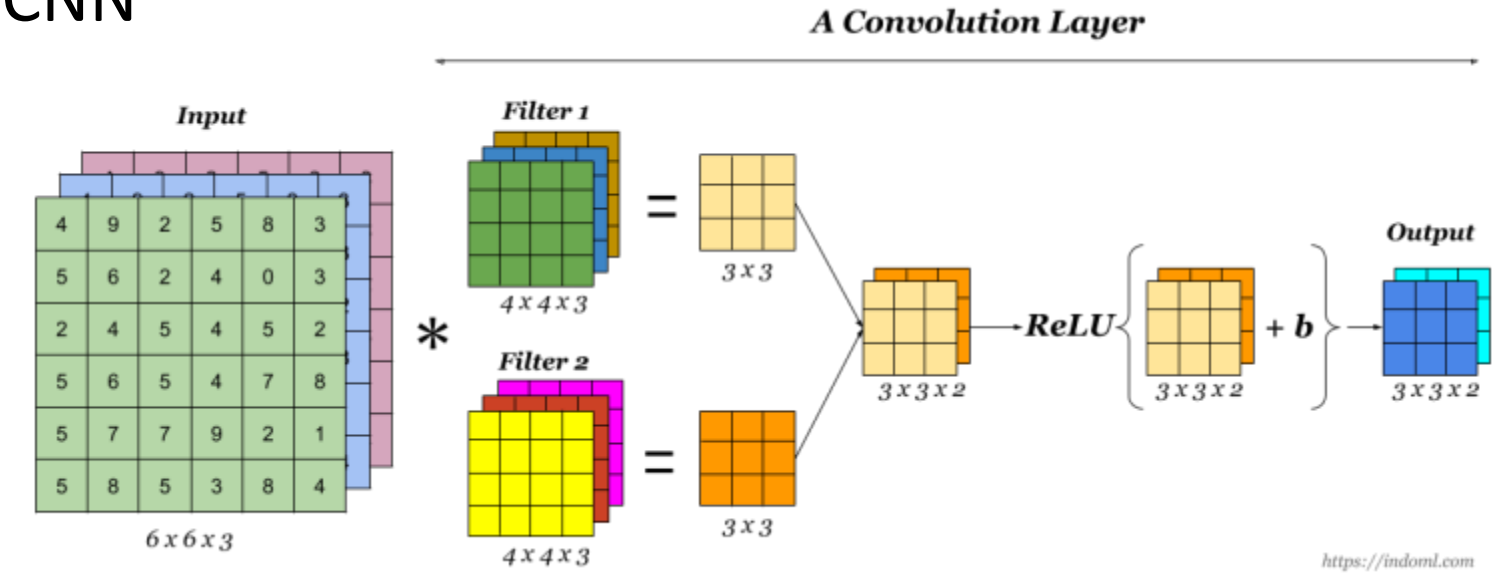
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



https://www.programmersought.com/article/35548177636/

A Convolutional Layer

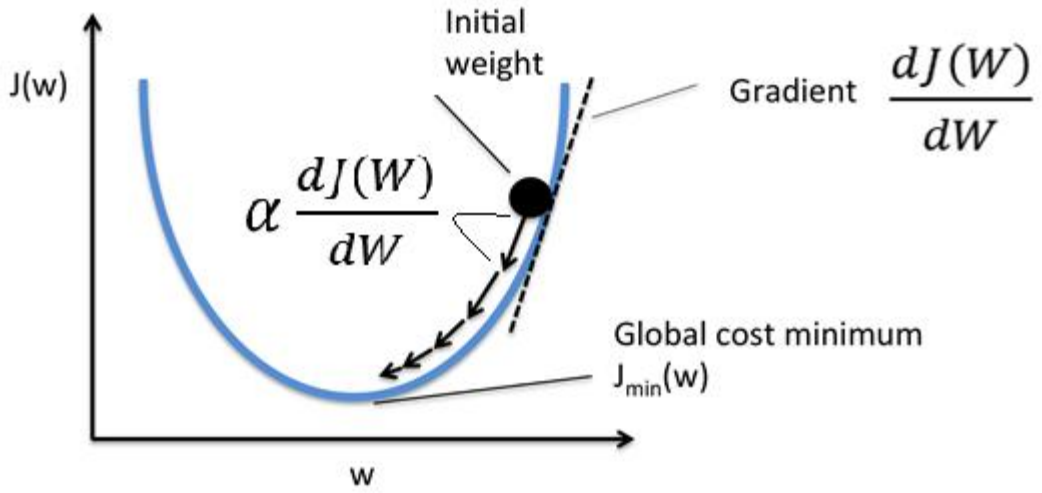
- A simple CNN



https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

Gradient Descent

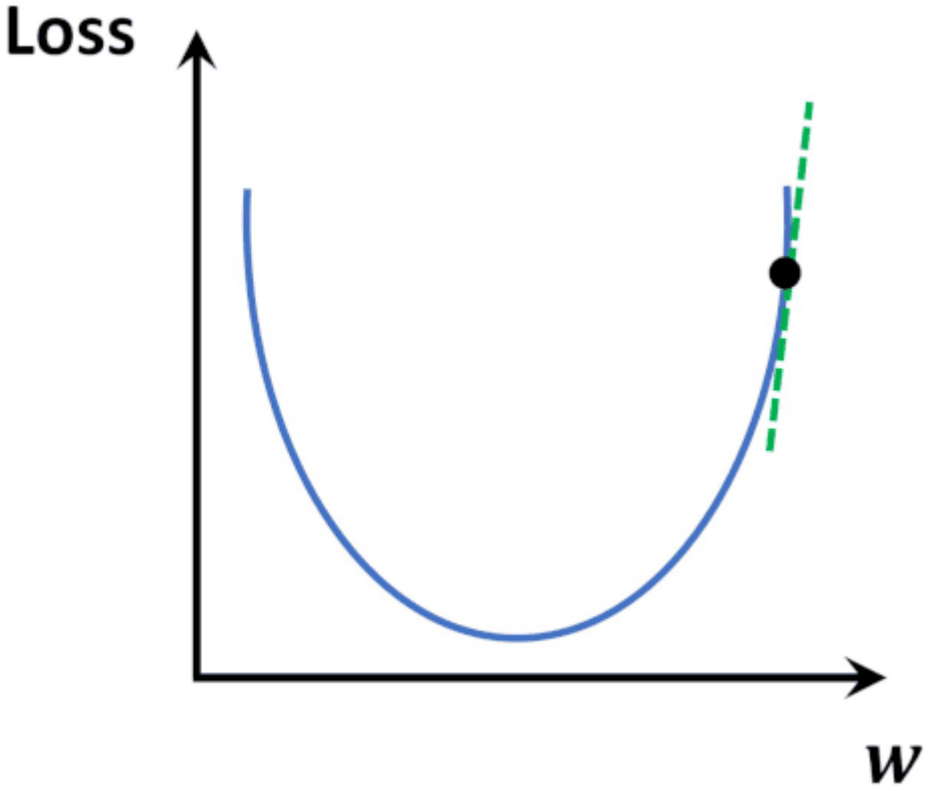
- To minimize J(W,b) we use Gradient Descent
- Gradient Descent is given by
- $W := W - \alpha \frac{dJ(W,b)}{dW}$
- $b := b - \alpha \frac{dJ(W,b)}{db}$



Gradient Descent

1. Initialization, select α
2. Calculate $dJ(\mathrm{W})/d\mathrm{W}$
3. Update the variable W
4. Repeat 2 and 3

- Question: How can we calculate the gradient in a deep network?

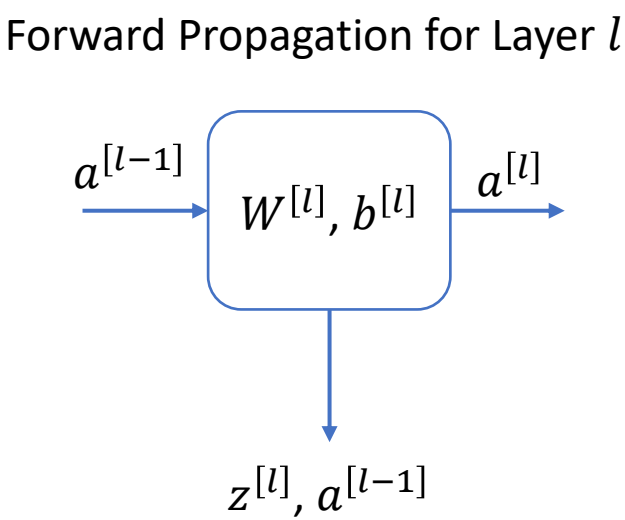


http://datahacker.rs/003-pytorch-how-to-implement-linear-regression-in-pytorch/

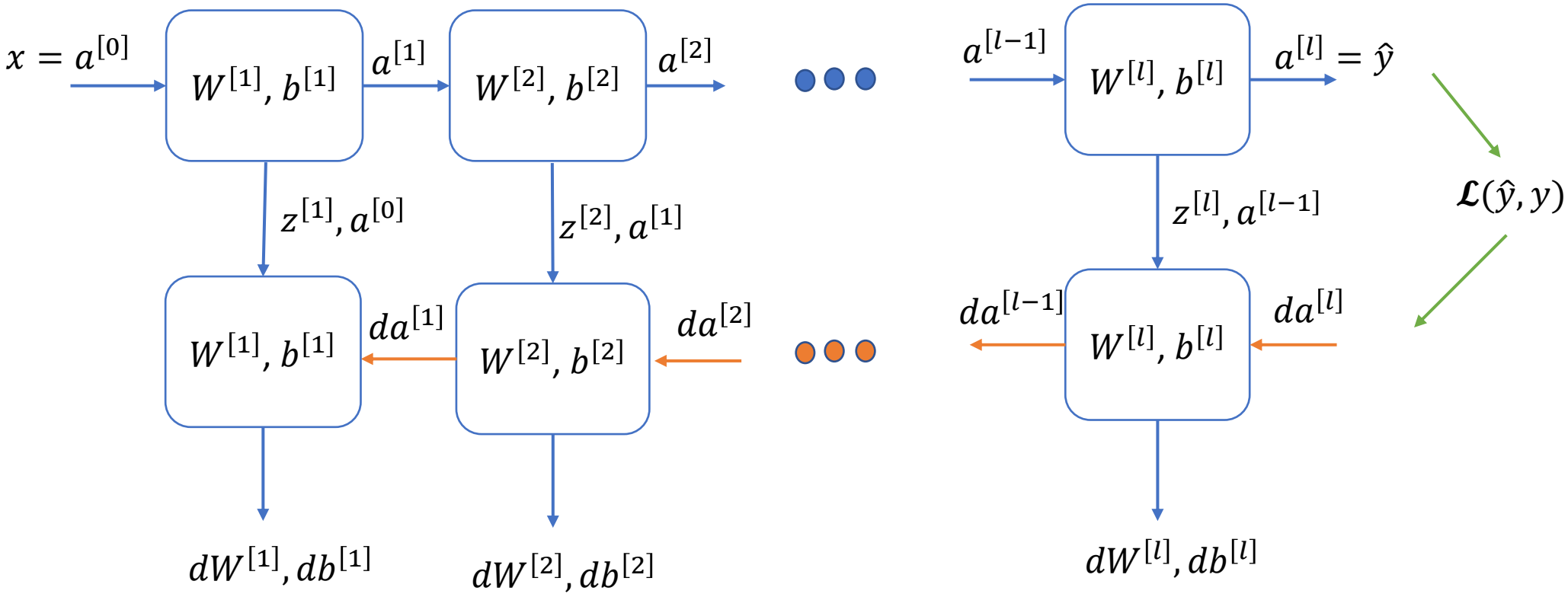
Forward Propagation for Layer l

- Input: $a^{[l-1]}$
- Output: $a^{[l]}$, Cache($z^{[l]}$, $a^{[l-1]}$)

1. $z^{[l]} = W^{[l]} * a^{[l-1]} + b^{[l]}$
2. $a^{[l]} = g^{[l]}(z^{[l]})$



Forward and Backward Propagation



Mini-Batch Gradient Descent

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{\mathcal{L}}\left(\hat{y}^{(i)}, y\right) \Rightarrow \frac{\partial J}{\partial W} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \boldsymbol{\mathcal{L}}}{\partial W}\left(\hat{y}^{(i)}, y\right), \frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial \boldsymbol{\mathcal{L}}}{\partial b}\left(\hat{y}^{(i)}, y\right)$$

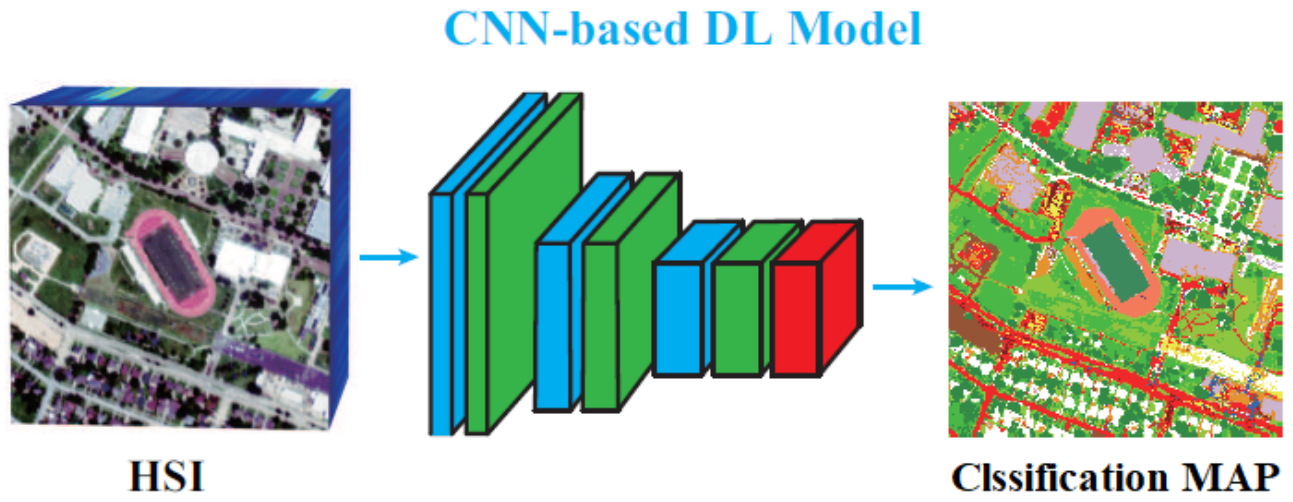
- Splitting the training set into n_2 mini batches

$$X = \left[x^{(1)}, \dots, x^{(n_1)} | x^{(n_1+1)}, \dots, x^{(2n_1)} |, \dots, x^{(m)}\right]$$

- For i=1: epoch
  - For j=1: n_2
    - Forward prop. on n_1 samples using vectorization
    - Calculate the cost
    - Backward prop. on n_1 samples using vectorization
    - Update (W, b)

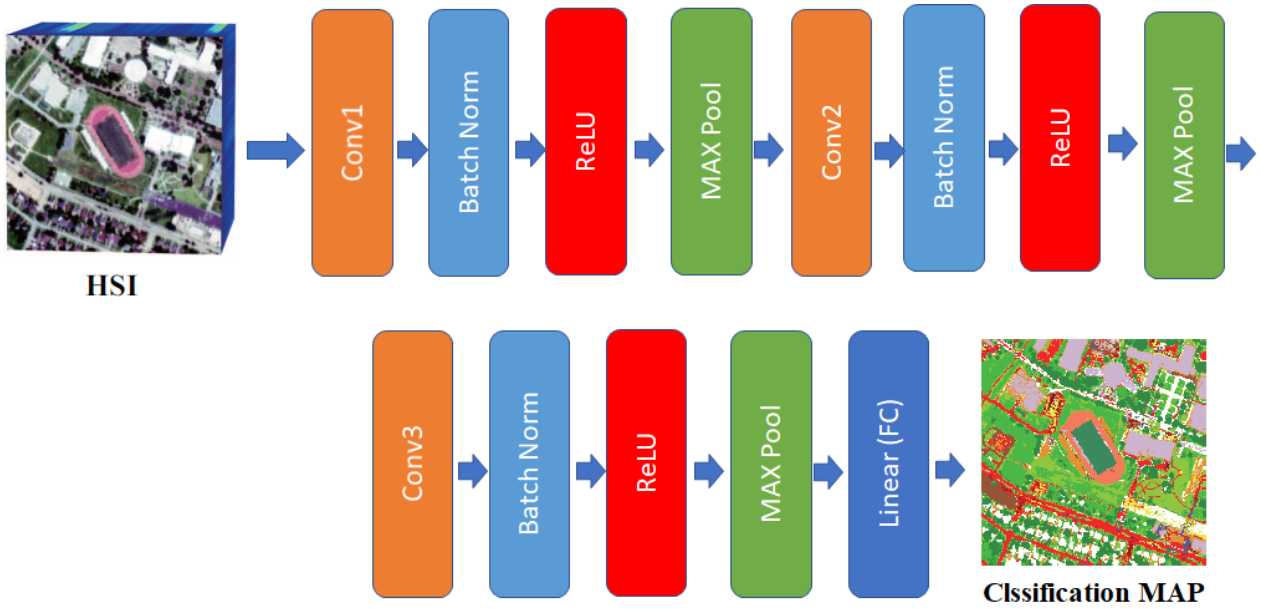
Convolutional Neural Network

- Main blocks: Convolutional layers, Batch normalization, Activation Function, (pooling layers) and fully connected layers



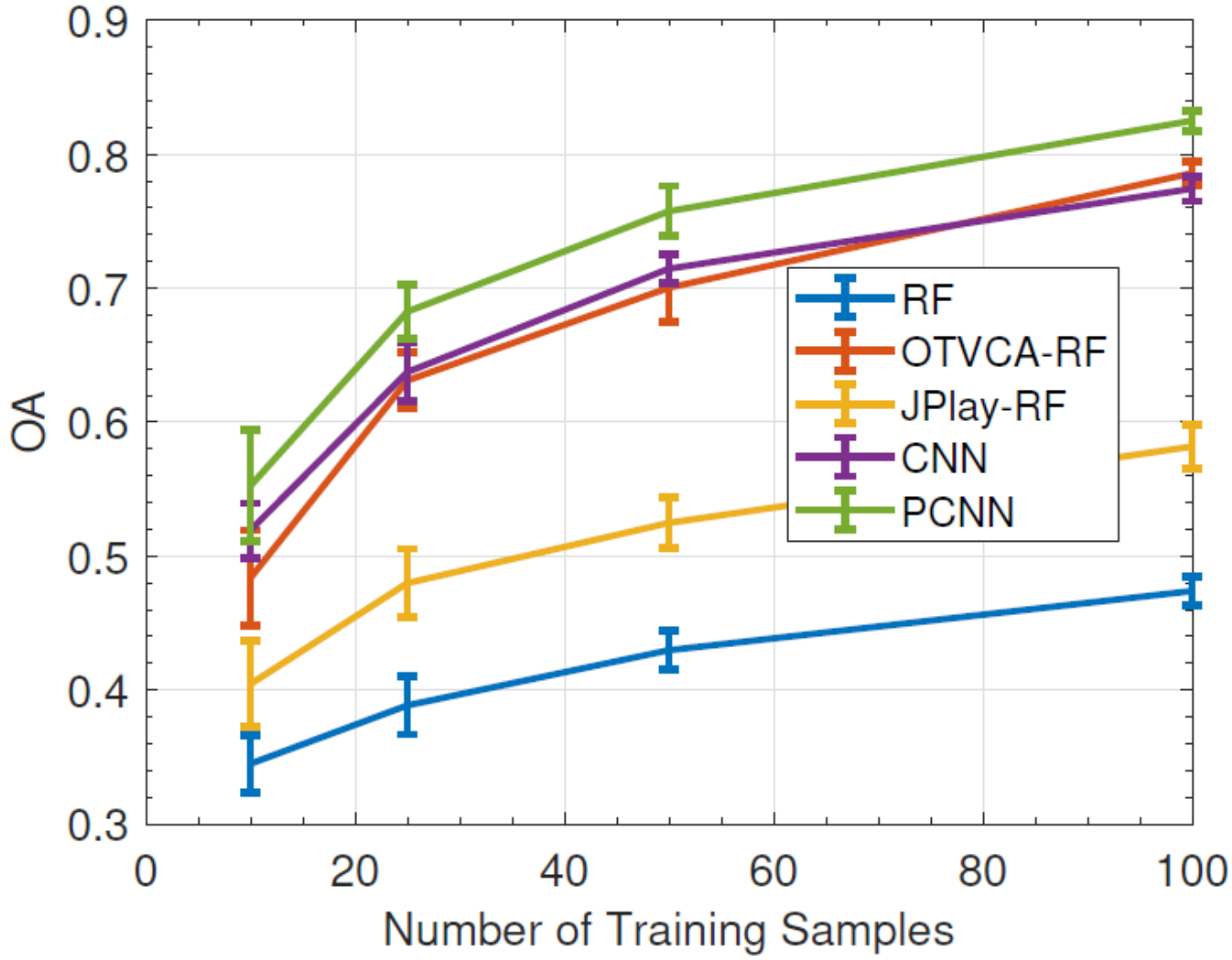
Convolutional Neural Network (CNN)

- The CNN Architecture used
- Number of filters: 32, 64, 128,respectively, Conv1, Conv2, Conv3
- Filter size 3x3
- Loss: Cross Entropy
- PCNN: PCA+CNN



Experimental Results: Feature Extraction

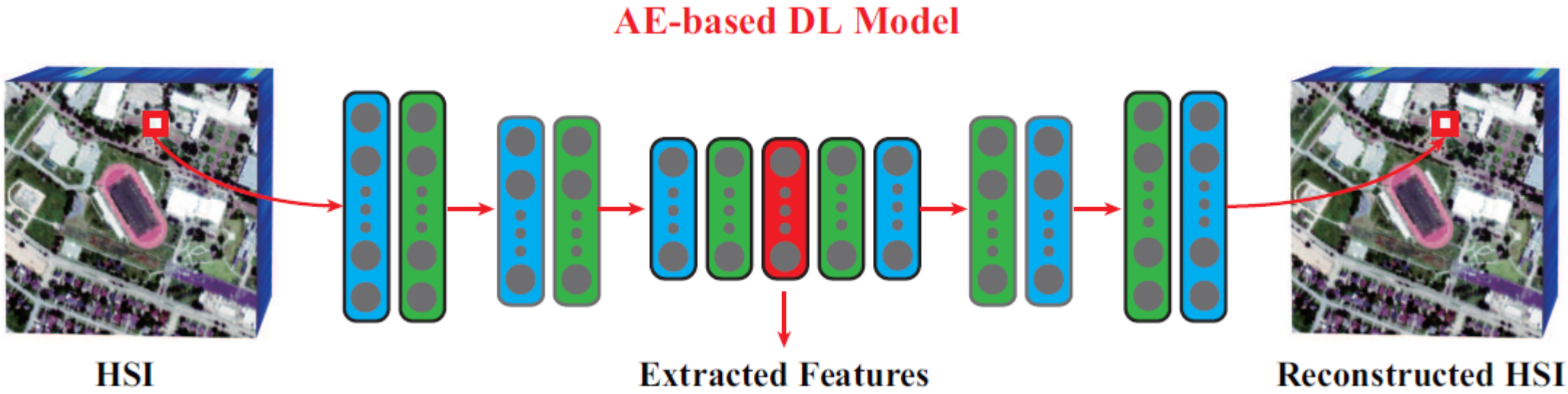
- Comparison in terms of OA w.r.t. different number of training samples (Houston Data)



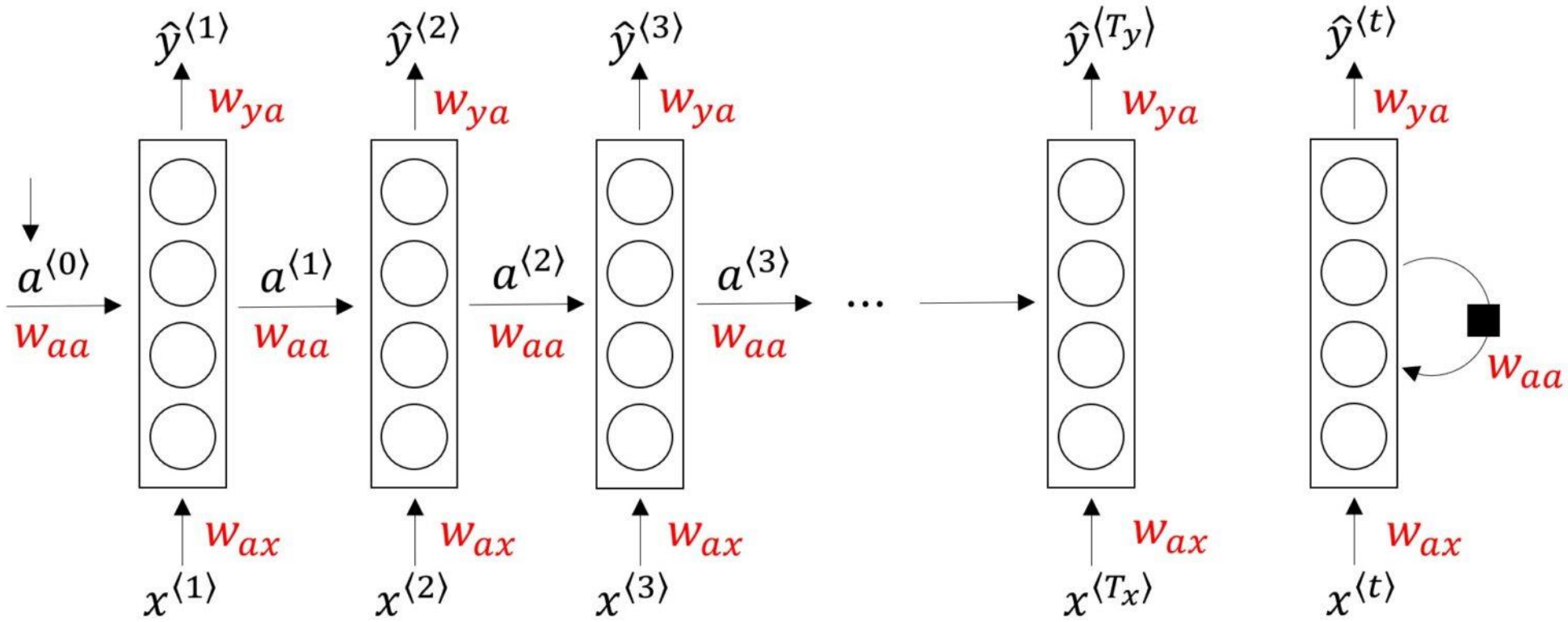
Autoencoders

- Loss: Reconstruction Error
- Classifier: Encoder

$$\mathbf{h} = f\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right),$$
$$\hat{\mathbf{x}} = f\left(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2\right),$$

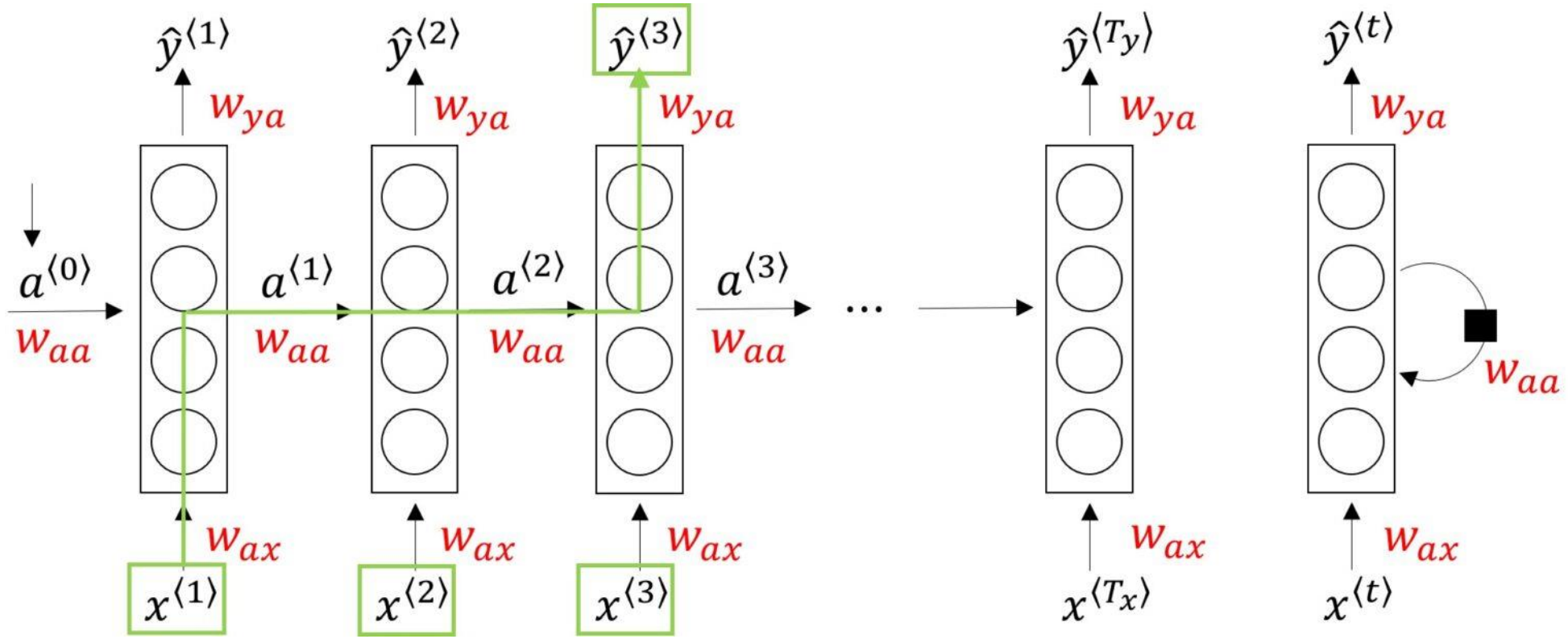


Recurrent Neural Networks (RNNs)



http://datahacker.rs/002-rnn-recurrent-neural-networks-architecture-mapping-and-propagation/

RNNs (Unidirectional)



http://datahacker.rs/002-rnn-recurrent-neural-networks-architecture-mapping-and-propagation/

RNNs

- Forward Propagation

$$a^{\langle t \rangle} = g\left(w_{aa} \times a^{\langle t-1 \rangle} + w_{ax} \times x^{\langle t \rangle} + b_a\right)$$

$$\hat{y}^{\langle t \rangle} = g\left(w_{ya} \times a^{\langle t \rangle} + b_y\right)$$

http://datahacker.rs/002-rnn-recurrent-neural-networks-architecture-mapping-and-propagation/

RNNs

- Forward Propagation

$$a^{\langle t \rangle} = g\left(\underbrace{w_{aa} \times a^{\langle t-1 \rangle} + w_{ax} \times x^{\langle t \rangle}}_{w_a} + b_a\right)$$

$$\overbrace{[\, w_{aa} \mid w_{ax} \,]}^{w_a} \times \begin{bmatrix} a^{\langle t-1 \rangle} \\ x^{\langle t \rangle} \end{bmatrix} = w_{aa} \times a^{\langle t-1 \rangle} + w_{ax} \times x^{\langle t \rangle}$$

$$a^{\langle t \rangle} = g\left(w_a \left[a^{\langle t-1 \rangle}, x^{\langle t \rangle} \right] + b_a \right)$$

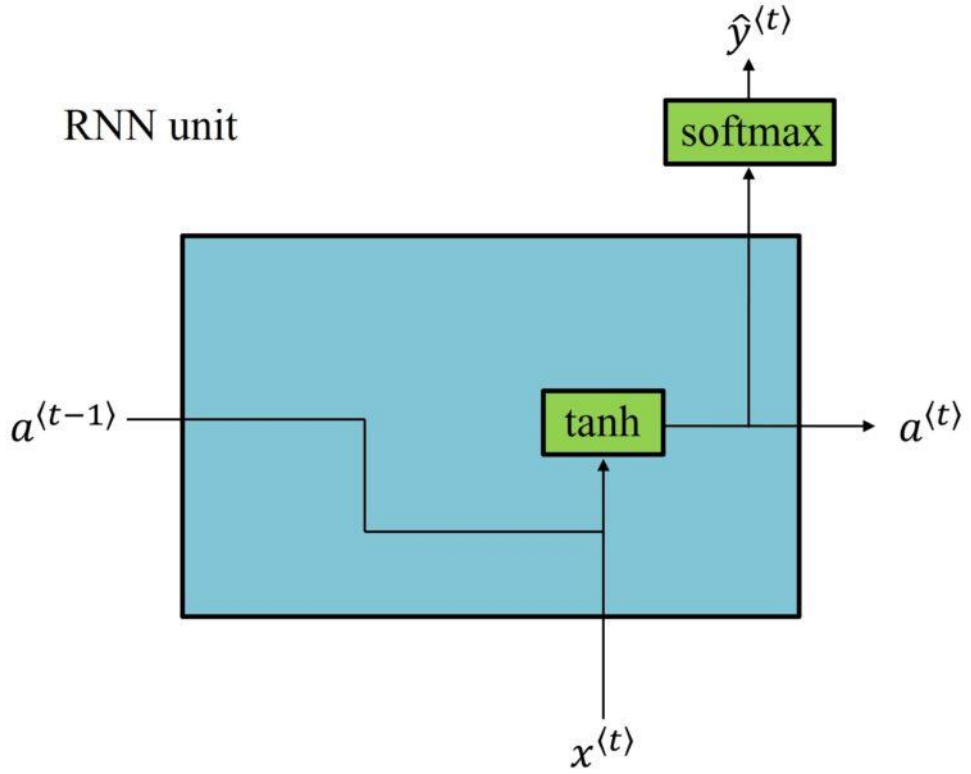
$$\hat{y}^{\langle t \rangle} = g\left(w_y \times a^{\langle t \rangle} + b_y\right)$$

http://datahacker.rs/002-rnn-recurrent-neural-networks-architecture-mapping-and-propagation/

RNN Unit

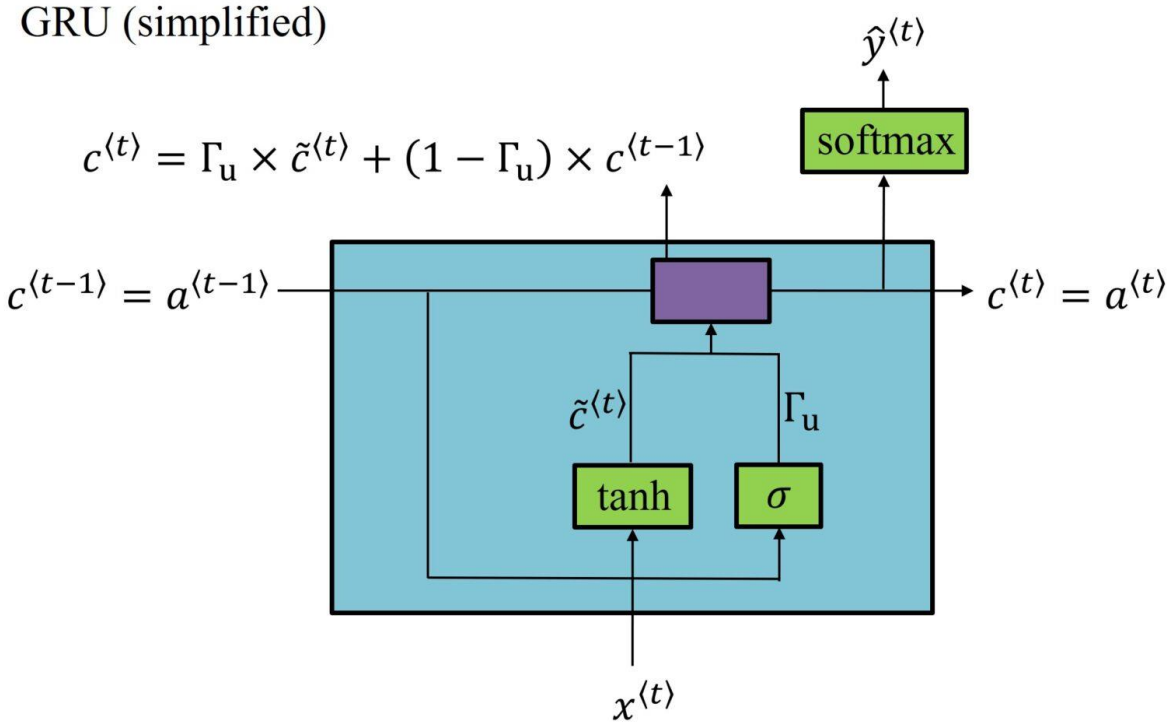
$$a^{\langle t \rangle} = g\left(w_a \left[a^{\langle t-1 \rangle}, x^{\langle t \rangle} \right] + b_a \right)$$

$$\hat{y}^{\langle t \rangle} = g\left(w_y \times a^{\langle t \rangle} + b_y \right)$$



http://datahacker.rs/005-rnn-tackling-vanishing-gradients-with-gru-and-lstm/

Gated Recurrent Unit (GRU)



$$\tilde{c}^{\langle t \rangle} = tanh\left(w_c \left[c^{\langle t-1 \rangle}, \quad x^{\langle t \rangle} \right] + b_c\right)$$

$$\Gamma_u = \sigma\left(w_u \left[c^{\langle t-1 \rangle}, \quad x^{\langle t \rangle} \right] + b_u\right)$$

$$c^{\langle t \rangle} = \Gamma_u \times \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) \times c^{\langle t-1 \rangle}$$

http://datahacker.rs/005-rnn-tackling-vanishing-gradients-with-gru-and-lstm/

Full GRU

$$\tilde{c}^{\langle t \rangle} = tanh\left(w_c \left[\Gamma_r \times c^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_c\right)$$

$$\Gamma_u = \sigma\left(w_u \left[c^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_u\right)$$

$$\Gamma_r = \sigma\left(w_r \left[c^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_r\right)$$

$$c^{\langle t \rangle} = \Gamma_u \times \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) \times c^{\langle t-1 \rangle}$$

http://datahacker.rs/005-rnn-tackling-vanishing-gradients-with-gru-and-lstm/

Long Short-Term Memory Unit (LSTM)

- LSTM

$$\tilde{c}^{\langle t \rangle} = tanh\left(w_c\left[a^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_c\right)$$
$$\Gamma_f = \sigma\left(w_f\left[a^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_f\right)$$
$$\Gamma_u = \sigma\left(w_u\left[a^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_u\right)$$
$$\Gamma_o = \sigma\left(w_o\left[a^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_o\right)$$
$$c^{\langle t \rangle} = \Gamma_u \times \tilde{c}^{\langle t \rangle} + \Gamma_f \times c^{\langle t-1 \rangle}$$
$$a^{\langle t \rangle} = \Gamma_o \times tanh\ c^{\langle t \rangle}$$

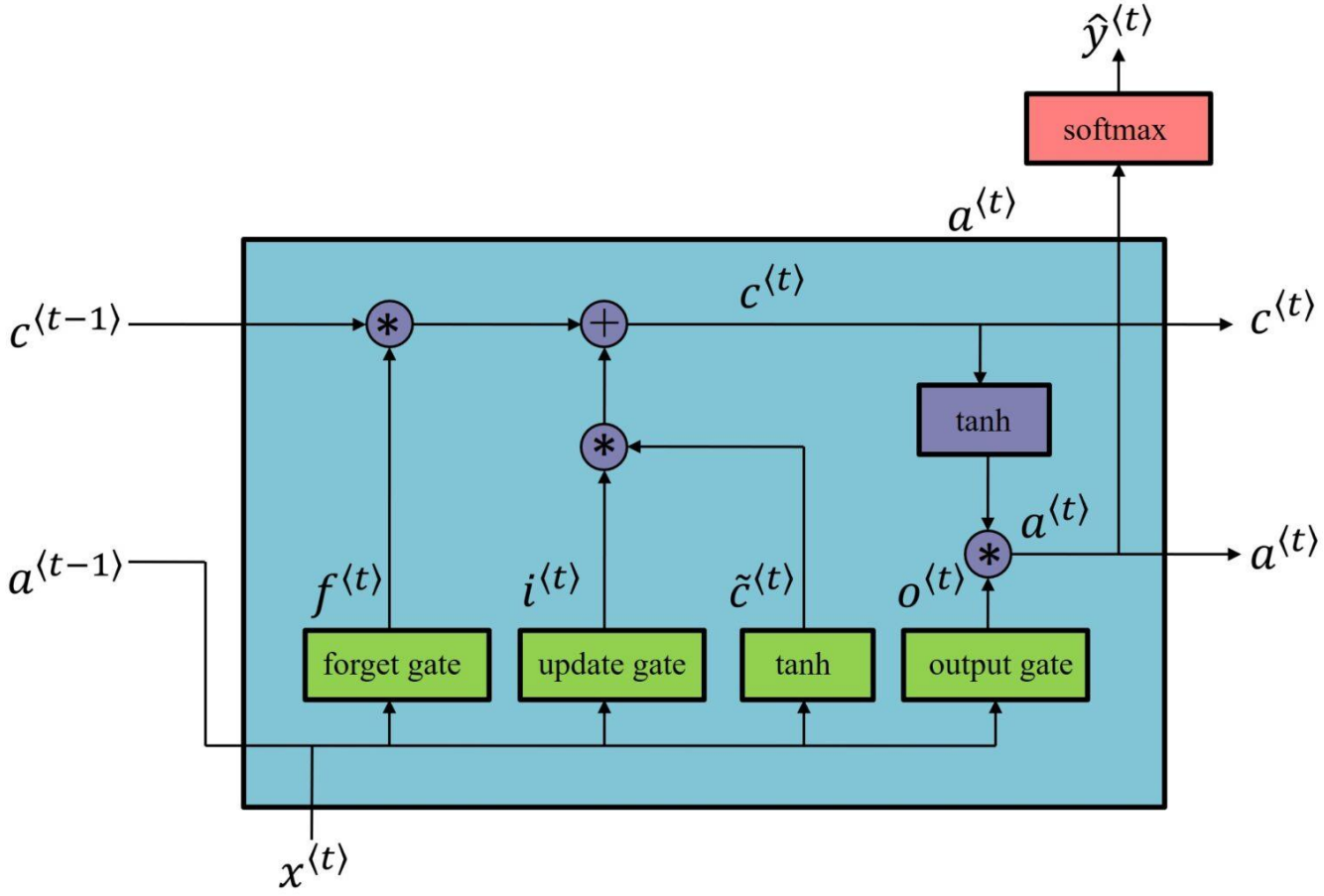
GRU

$$\tilde{c}^{\langle t \rangle} = tanh\left(w_c\left[\Gamma_r \times c^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_c\right)$$
$$\Gamma_r = \sigma\left(w_r\left[c^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_r\right)$$
$$\Gamma_u = \sigma\left(w_u\left[c^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_u\right)$$
$$c^{\langle t \rangle} = \Gamma_u \times \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) \times c^{\langle t-1 \rangle}$$
$$a^{\langle t \rangle} = c^{\langle t \rangle}$$

http://datahacker.rs/005-rnn-tackling-vanishing-gradients-with-gru-and-lstm/

LSTM

$$\tilde{c}^{\langle t \rangle} = tanh\left(w_c\left[a^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_c\right)$$
$$\Gamma_f = \sigma\left(w_f\left[a^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_f\right)$$
$$\Gamma_u = \sigma\left(w_u\left[a^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_u\right)$$
$$\Gamma_o = \sigma\left(w_o\left[a^{\langle t-1 \rangle}, \quad x^{\langle t \rangle}\right] + b_o\right)$$
$$c^{\langle t \rangle} = \Gamma_u \times \tilde{c}^{\langle t \rangle} + \Gamma_f \times c^{\langle t-1 \rangle}$$
$$a^{\langle t \rangle} = \Gamma_o \times tanh\ c^{\langle t \rangle}$$



http://datahacker.rs/005-rnn-tackling-vanishing-gradients-with-gru-and-lstm/

RNN vs. GRU vs. LSTM

RNN cannot learn the long-term dependencies, why?

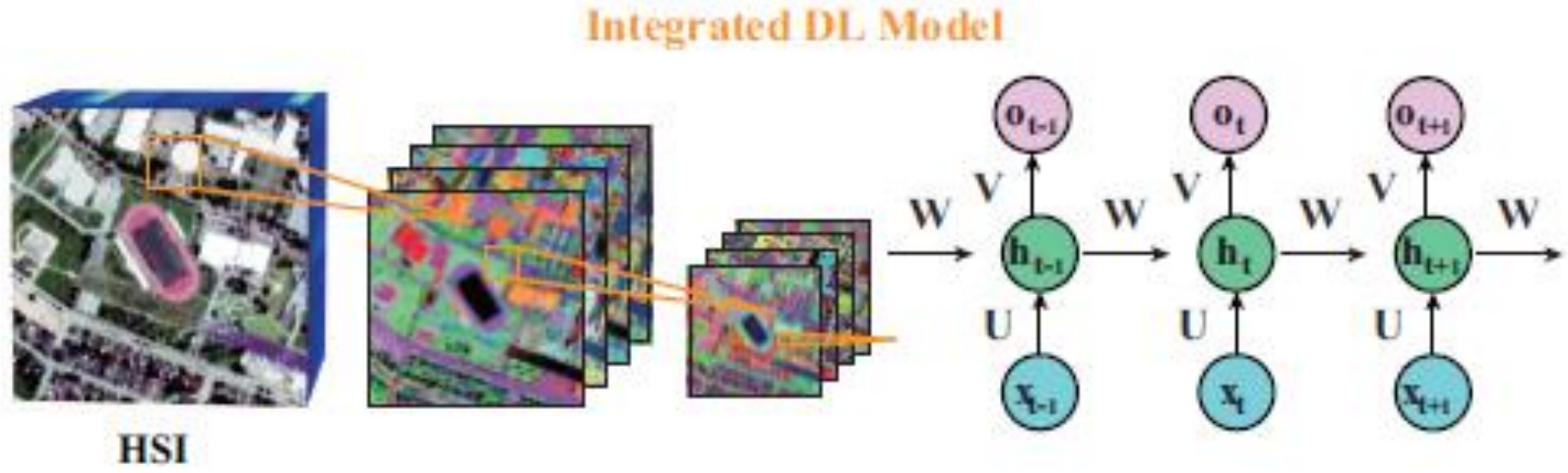
Both GRU and LSTM are good to capture the long-term dependencies!

GRU is simple and fast

LSTM is more complex and might be more effective

Integrated Networks

- AEs and RNNs are good at processing vectorized inputs and extracting spectral features
- CNNs are more powerful in extracing spatial features
- Integrated networks aim to exploit both advantages



Experimental Setup

AE: Three hidden layers with 32, 64, and 128 neurons, and ReLU activation function

RNN: Two recurrent layers with GRU each with 128 neurons

Convolutional AE (CAE): Three convolutional layers with 32, 64, and 128 filters (3x3) and three de-convolutional layers with 64, 32, and p

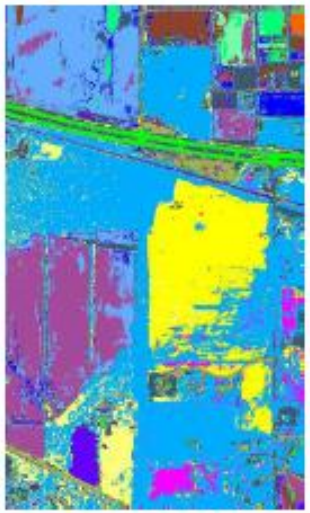
Convolutional RNN (CRNN): two recurrent layers with convolutional LSTM units and 32, 64, and 128 filters (3x3).

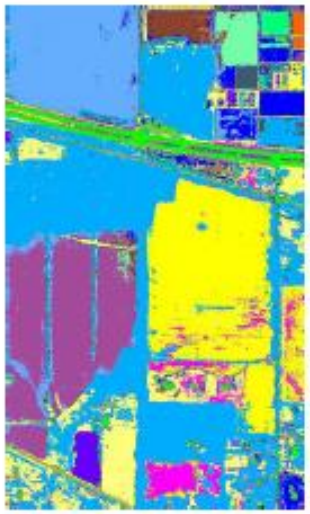
Classification Accuracies

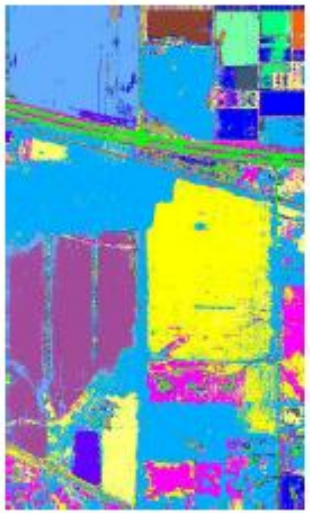
		Shallow FE								Deep FE					
		UFE				SFE				SFE					
	Spectral	PCA	MSTV	OTVCA	LPP	LDA	CGDA	LSDR	JPlay	AE	RNN	CNN	CAE	CRNN	PCNN
Indian Pines 2010															
AA	0.7465	0.8043	0.8428	0.8194	**0.8532**	0.8341	0.7861	0.8089	0.8367	0.8197	0.7737	0.8480	0.8478	0.8233	0.8498
OA	0.7866	0.8598	0.8561	0.8378	**0.9112**	0.8748	0.8370	0.8748	0.8829	0.8836	0.8655	0.8945	0.8911	0.8525	0.9018
κ	0.7390	0.8297	0.8247	0.8054	**0.8909**	0.8481	0.8010	0.8466	0.8571	0.8580	0.8355	0.8716	0.8673	0.8213	0.8802
Houston 2013															
AA	0.7679	0.8371	0.8347	**0.8901**	0.8239	0.8056	0.8094	0.7878	0.8532	0.7716	0.7976	0.8388	0.8350	0.8210	0.8664
OA	0.7278	0.8058	0.8088	**0.8753**	0.7874	0.7745	0.7789	0.7524	0.8280	0.7436	0.7646	0.8239	0.8184	0.7921	0.8526
κ	0.7076	0.7895	0.7923	**0.8648**	0.7700	0.7552	0.7604	0.7315	0.8134	0.7235	0.7469	0.8096	0.8036	0.7761	0.8404
Houston 2018															
AA	0.5195	0.6039	0.5707	0.5696	0.6060	0.5952	0.5676	0.5647	0.5863	0.5777	0.5458	0.6400	0.6268	0.6090	**0.6667**
OA	0.4634	0.5101	0.5750	0.5899	0.5552	0.5027	0.4825	0.5492	0.5944	0.5938	0.4851	0.7278	0.6969	0.5116	**0.7728**
κ	0.3732	0.4317	0.4833	0.4974	0.4714	0.4231	0.4018	0.4560	0.5037	0.4948	0.3936	0.6474	0.6124	0.4372	**0.7011**

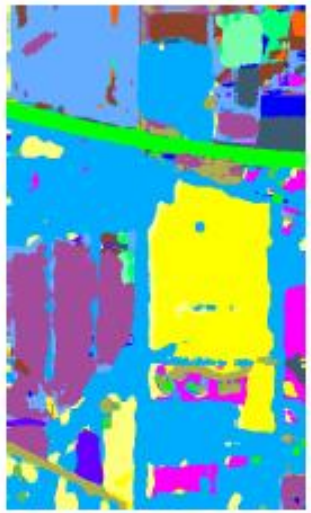
Experimental Results: Indian Pines 2010

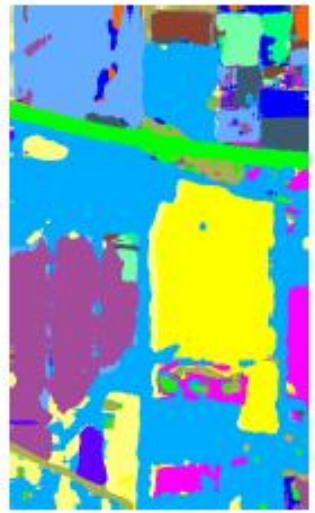
Comparisons of classification maps


(a) HSI


(b) LPP


(c) JPlay


(d) CNN


(e) PCNN


(f) Ground Reference

Experimental Results: Houston 2013

Comparisons of classification maps

(a) HSI

(b) OTVCA

(c) JPLAY

(d) CNN

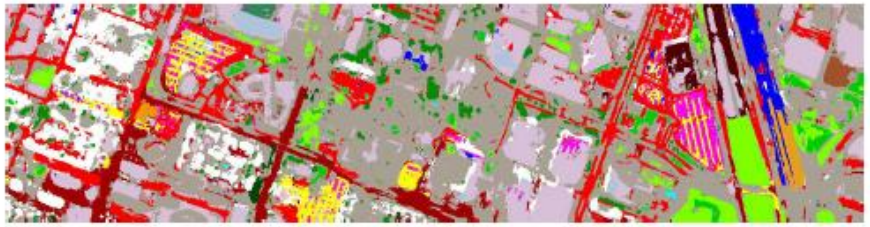
(e) PCNN

(f) Ground Truth

Experimental Results: Houston 2018

Comparisons of classification maps

(a) HSI

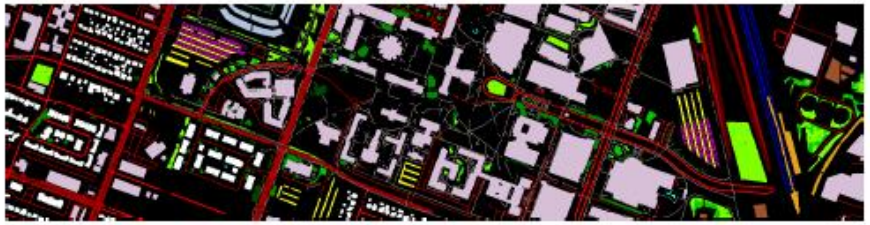


(b) OTVCA

(c) JPlay

(d) CNN

(e) PCNN



(f) Ground Reference

Conclusion

The limited number of training samples and the selection of hyperparameters are of main challenges for DL-based HSI analysis

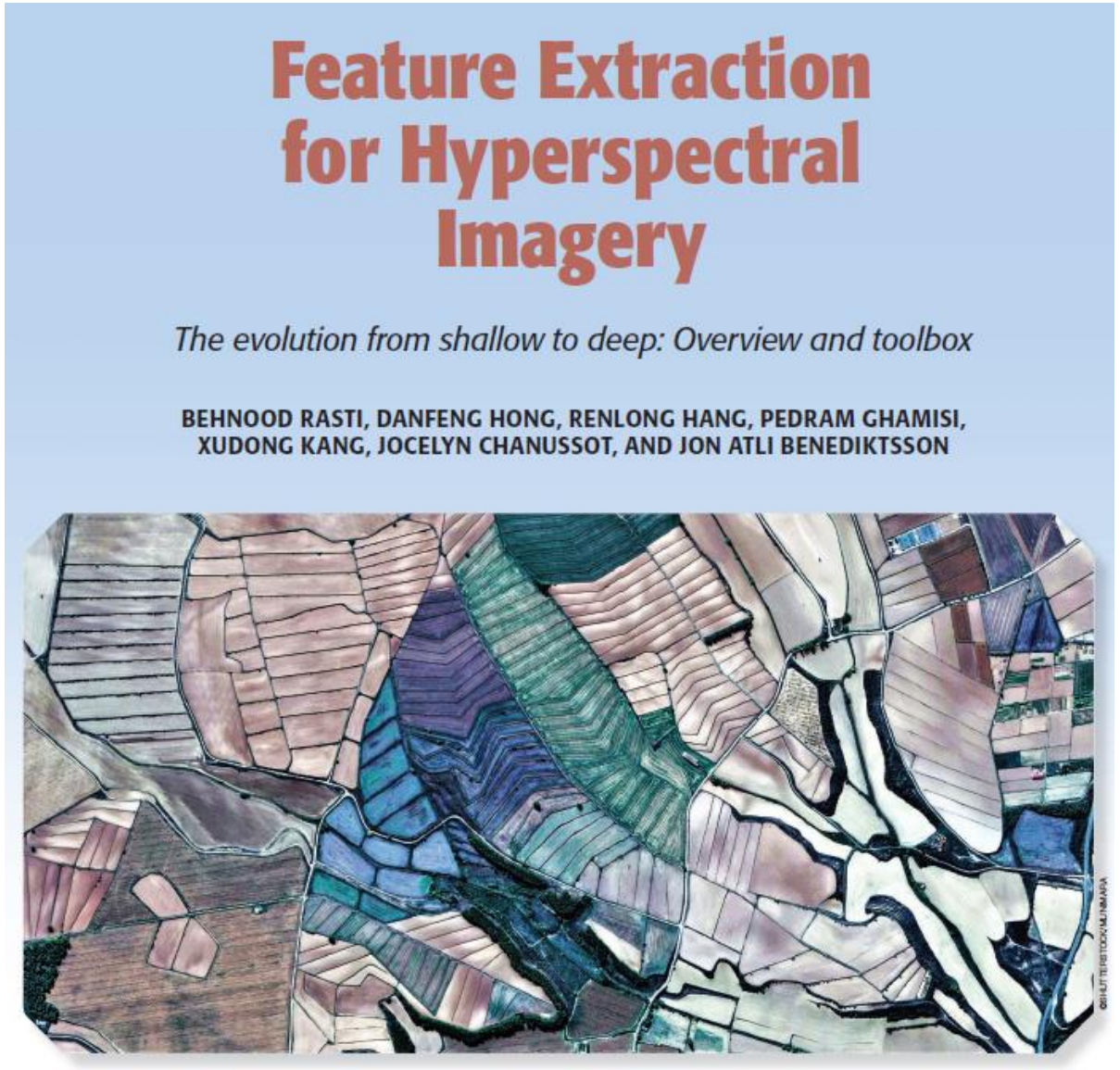
UFE outperforms SFE in terms of classification accuracy for supervised classification

The basic end-to-end deep feature extraction techniques cannot considerably outperform the shallow feature extraction techniques for hyperspectral classification

Hybrid models show superiority to both deep and shallow feature extraction techniques

Hyperspectral-Shallow-Deep-Feature-Extraction-Toolbox (HyFTech)

- The toolbox is available:
- https://github.com/BehnoodRasti/HyFTech-Hyperspectral-Shallow-Deep-Feature-Extraction-Toolbox



Unterstützt von / Supported by

Alexander von Humboldt
Stiftung/Foundation

HIF
HELMHOLTZ INSTITUTE FREIBERG
FOR RESOURCE TECHNOLOGY

HZDR

HELMHOLTZ
ZENTRUM DRESDEN
ROSSENDORF

Appreciation

- Prof. Jon Atli Benediktsson
- Prof. Jocelyn Chanussot
- Prof. Xudong Kang
- Prof. Pedram Ghamisi
- Prof. Renlong Hang
- Dr. Danfeng Hong