Machine Learning Techniques for Radar Automatic Target Recognition (ATR)

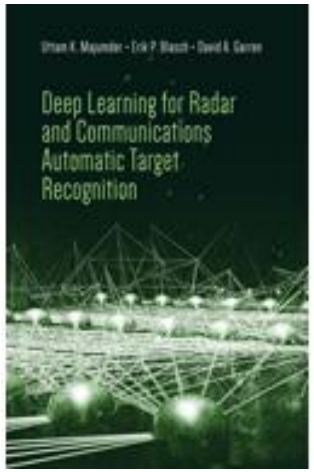
Uttam K. Majumder

Chapter 4: Deep Learning

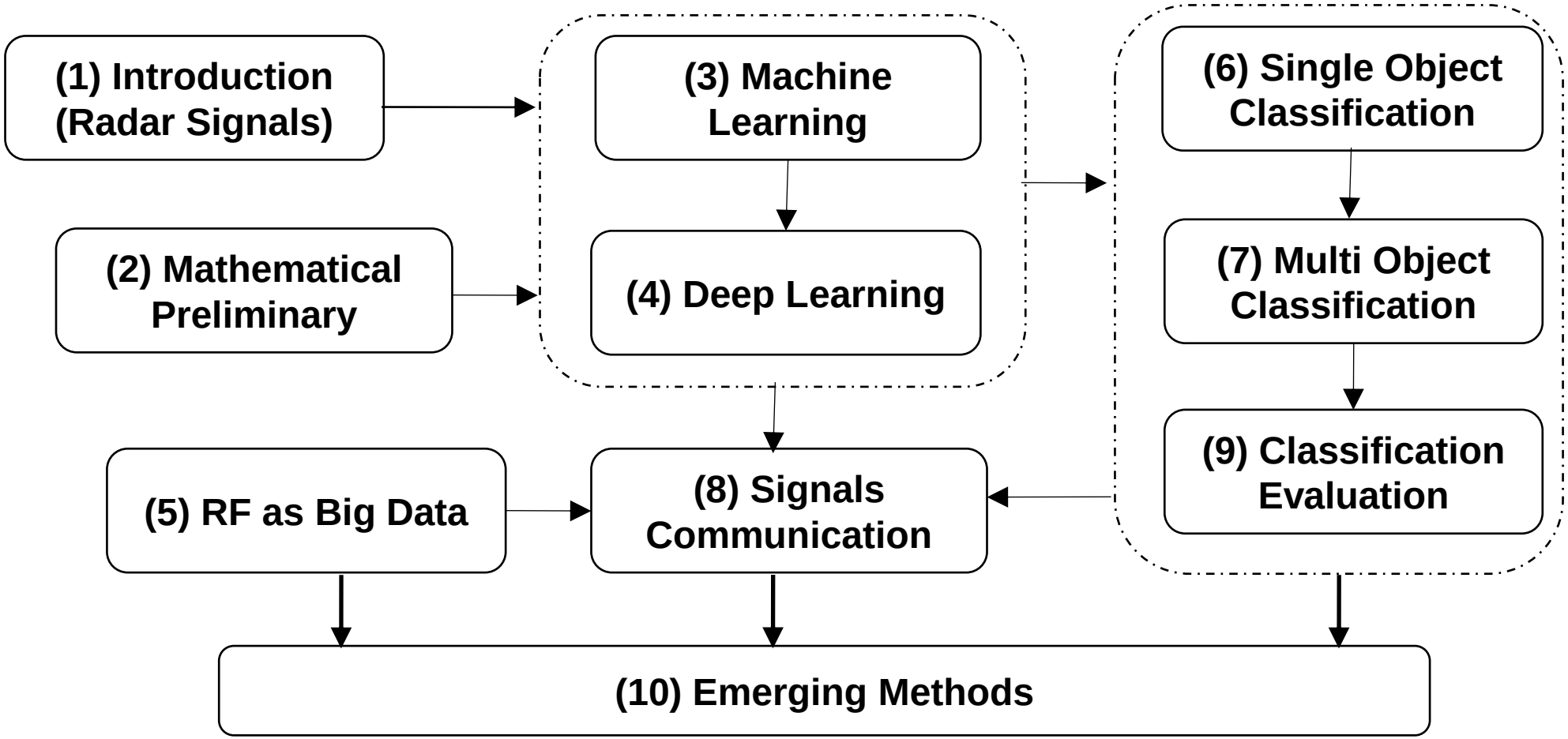
U. Majumder, E. Blasch, D. Garren, *Deep Learning for Radar and Communications Automatic Target Recognition*, Artech House, 2020.

Lecture Outline

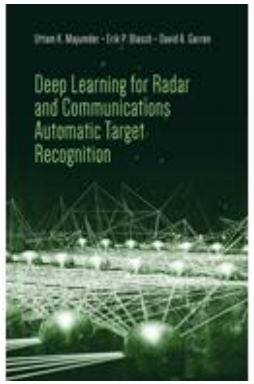
1. Radio Frequency ATR: Past, Present, and Future:
   20 min
2. Mathematics for Machine Learning / Deep Learning:
   20 min
3. Review of ML Algorithms: 25 min
4. **Deep Learning Algorithms: 30 min**
5. RF Data for ML Research: 15 min
6. DL for Single Target Classification: 25 min
7. DL for Many Targets Classification: 25 min
8. RF Signals Classification: 20 min
9. RF ATR Performance Evaluation: 25 min
10. Emerging ML Algorithms for RF ATR: 35 min



Deep Learning (DL) Based Radio Frequency (RF) Automatic Target Recognition (ATR)

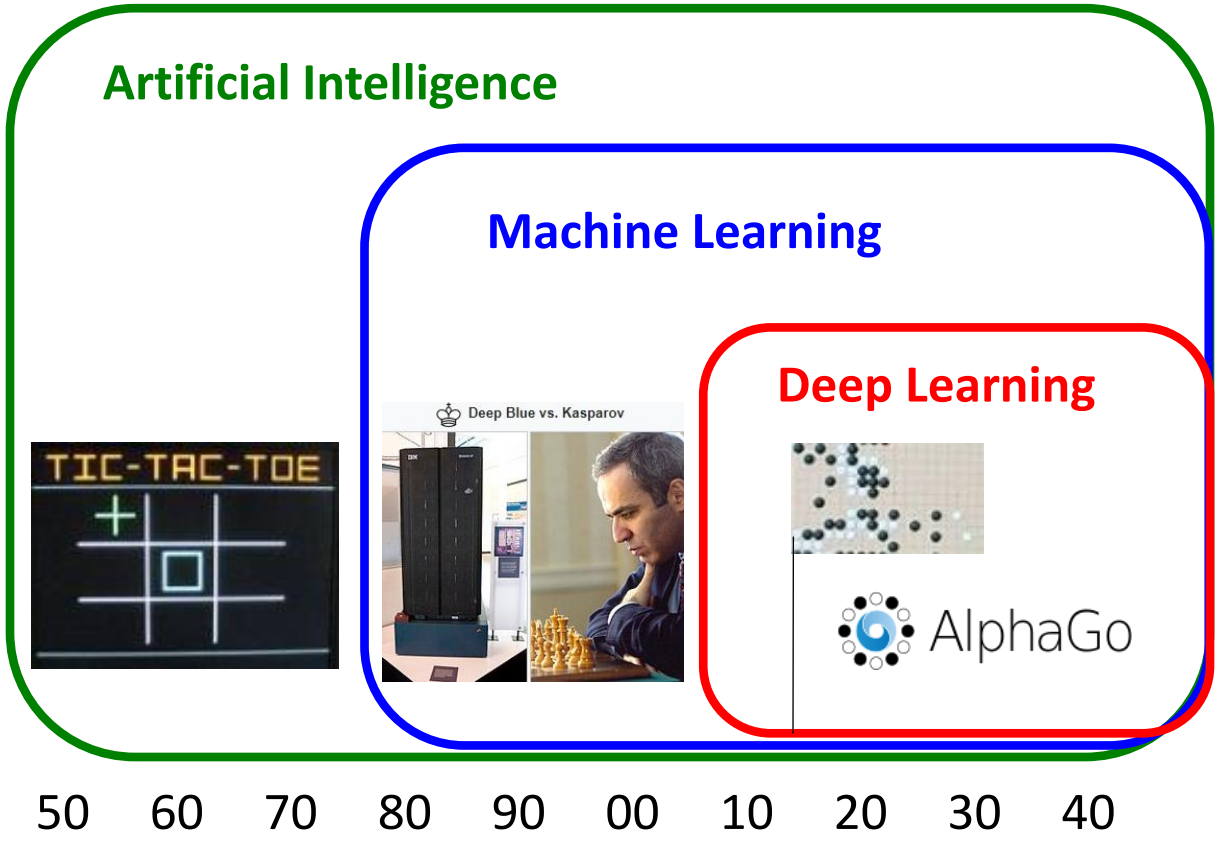


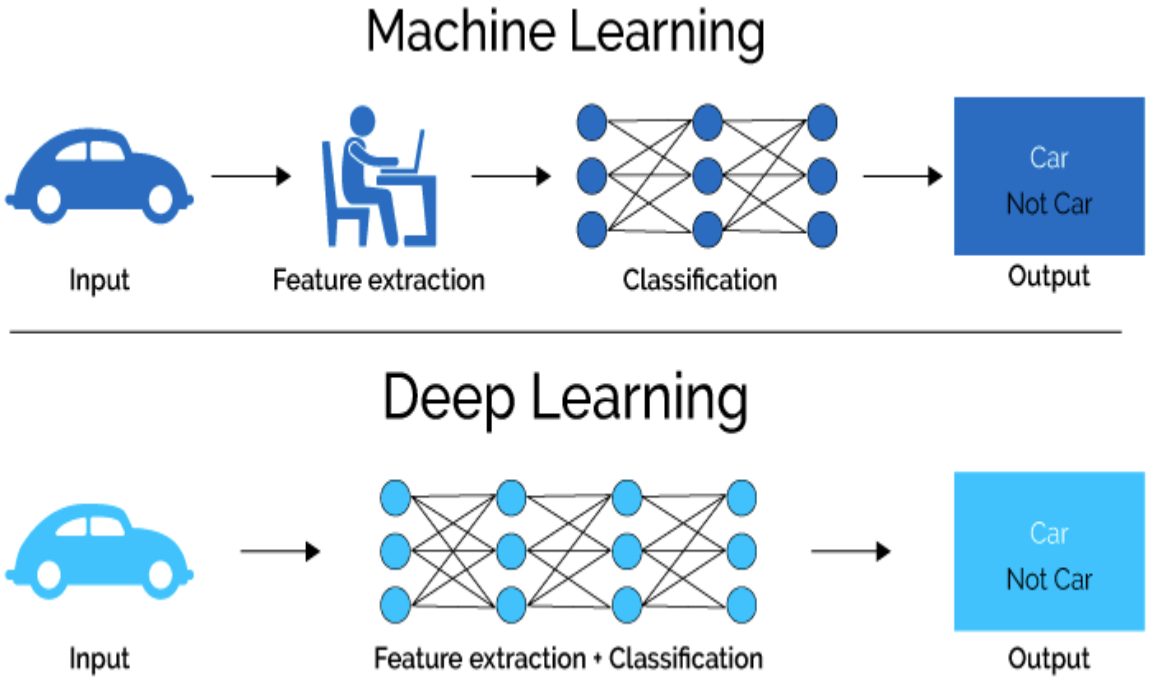
Chapter 4 Outline



Deep Learning

- Deep Learning – took off in 2012 from the CVPR competition (Univ. Toronto)





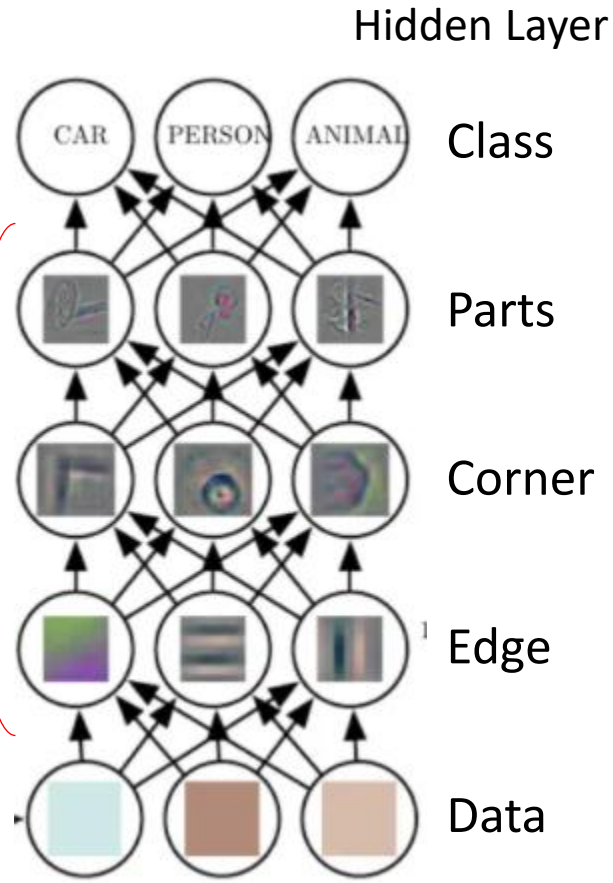
[PLATAFORMA] Inteligencia Artificial: ML, DL +Data Vol.I - ForoIngenieros

Deep Learning

- 1986 – Restricted Boltzman Machine (Smolensky)
- 1986 - Recurrent Neural Networks (Jordan)
- 1986 – Multilayer Perceptron, Autoencoder (Rumelhart, Hinton, Williams)
- 1987 – Bidirectional RNN (Schuster and Paliwal)
- 1988 – Convolutional Neural Network, LeNet (Lecun)

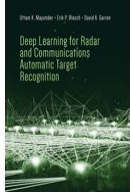
- 1997 – Long Short-Term Memory (Hochreiter and Schmidhuber)
- 2006 – **Deep Belief (Neural) Networks (Hinton)**

- 2012 – **DNN wins Image Competition**
- 2012 – Deep Boltzmann Machines (Salakhutdinov and Hinton)
- 2014 – Generative Adversarial Networks (Goodfellow)
- 2017 – Capsule Networks (Sabour, Frost, Hinton)
- Transfer learning, Zero-Shot Learning



What are convolutional neural networks (CNN)? – TechTalks (bdtechtalks.com)

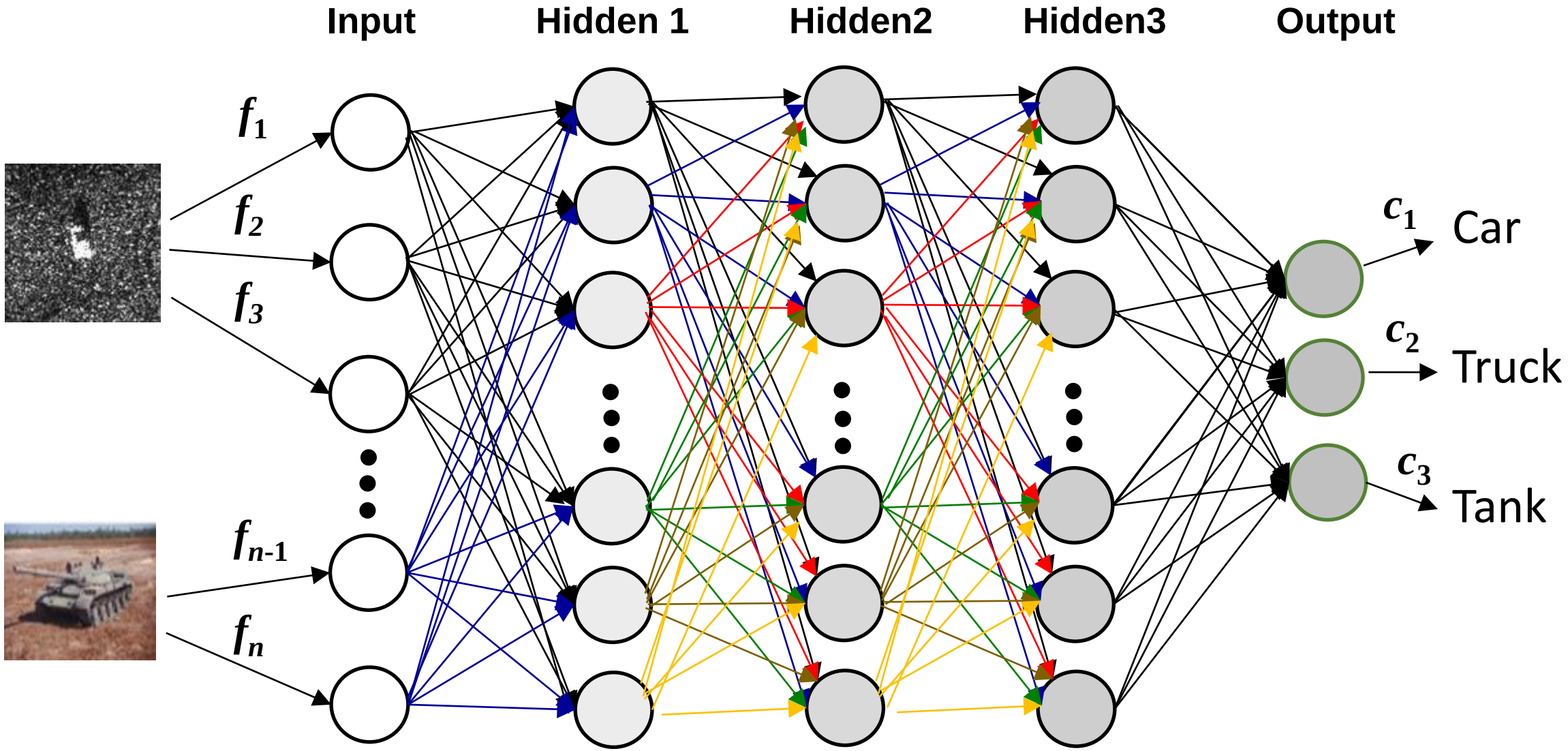
Takeaway Message



Method	Performance (%)			Reference
	SOC	EOC-1	EOC-2	
Pulse Couple Neural Network (PCNN) [35 tgts]	87.0	75	67	Blasch, 1998
A-ConvNets	99.1	96.1	98.9	Chen, *et al*., 2016
AFRLeNet	**99.4**			**Profeta *et al*., 2016**
CNN-SVM	99.5	95.75		Amrani, *et al*., 2017
VGG-S1	98.8	94.15		Amrani, *et al*., 2017
VGG-S, Feature Fusion, KNN	99.82	96.16		Amrani, *et al*., 2017
Multi-aspect aware bidirectional LSTM RNN	99.90	-	99.59	Zhang, *et al*. 2017
Joint Supervised Dictionary	97.65	98.39		
GAN-CNN	97.53			Zheng, C et al, 2019
MGAN-CNN	97.81			Zheng, C et al, 2019
Triple GAN and Integrated GAN	99.9			Gao, F. Et al 2019

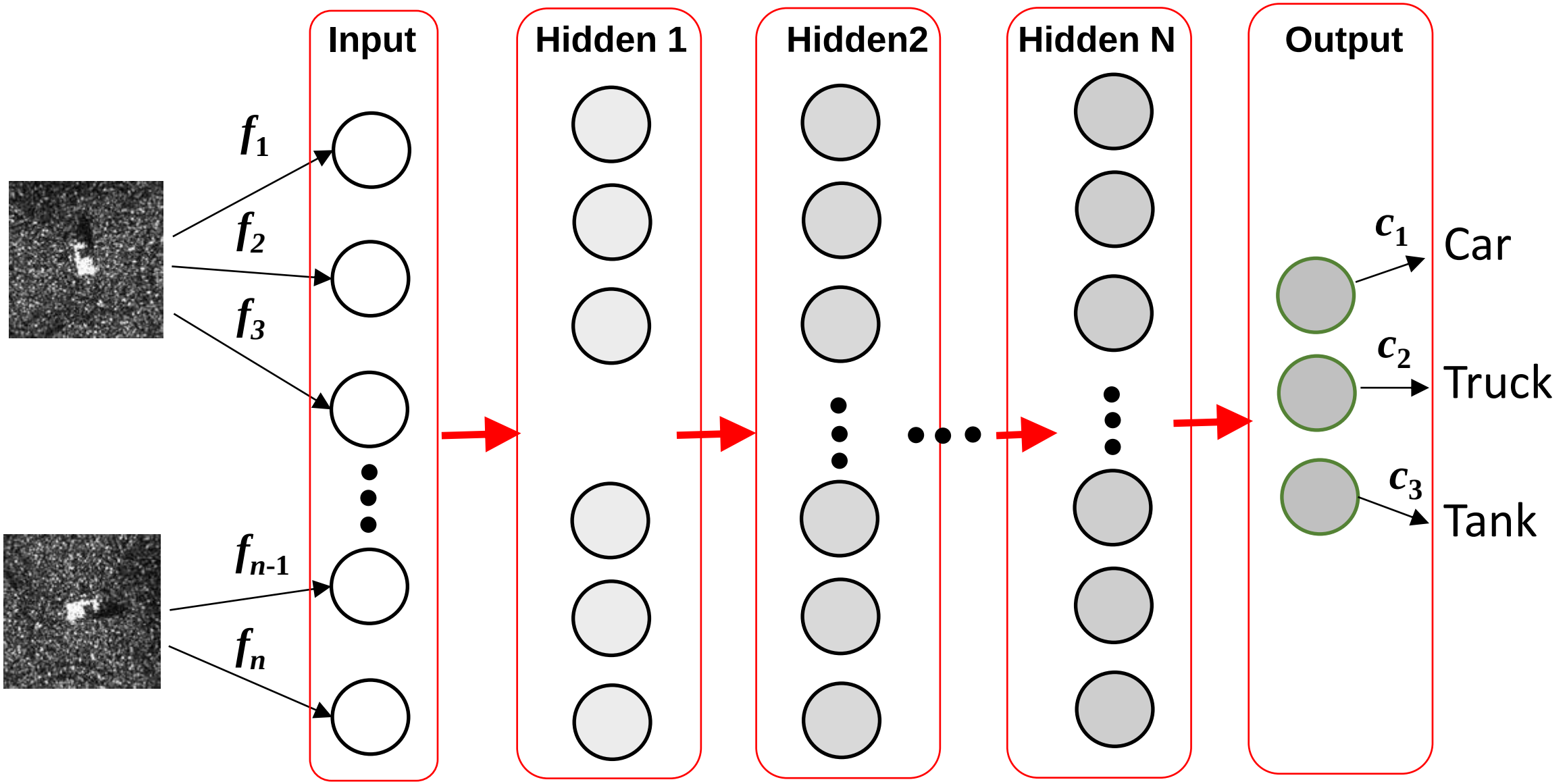
DNN Model (Connected)

Figure 4.1



DNN Model (Hidden Layers)

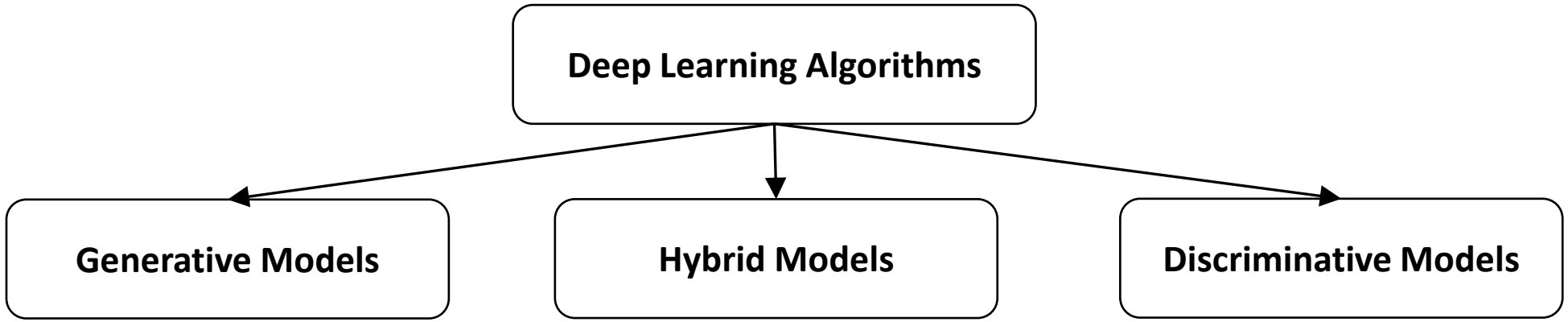
Figure 4.2



Types of Deep Learning

Figure 4.3

Generative – form features from low-dimensional space
Discriminative – determine important features among choices



- Deep Belief Networks (DBN)
- Deep Boltzmann Machine
- **Autoencoder**

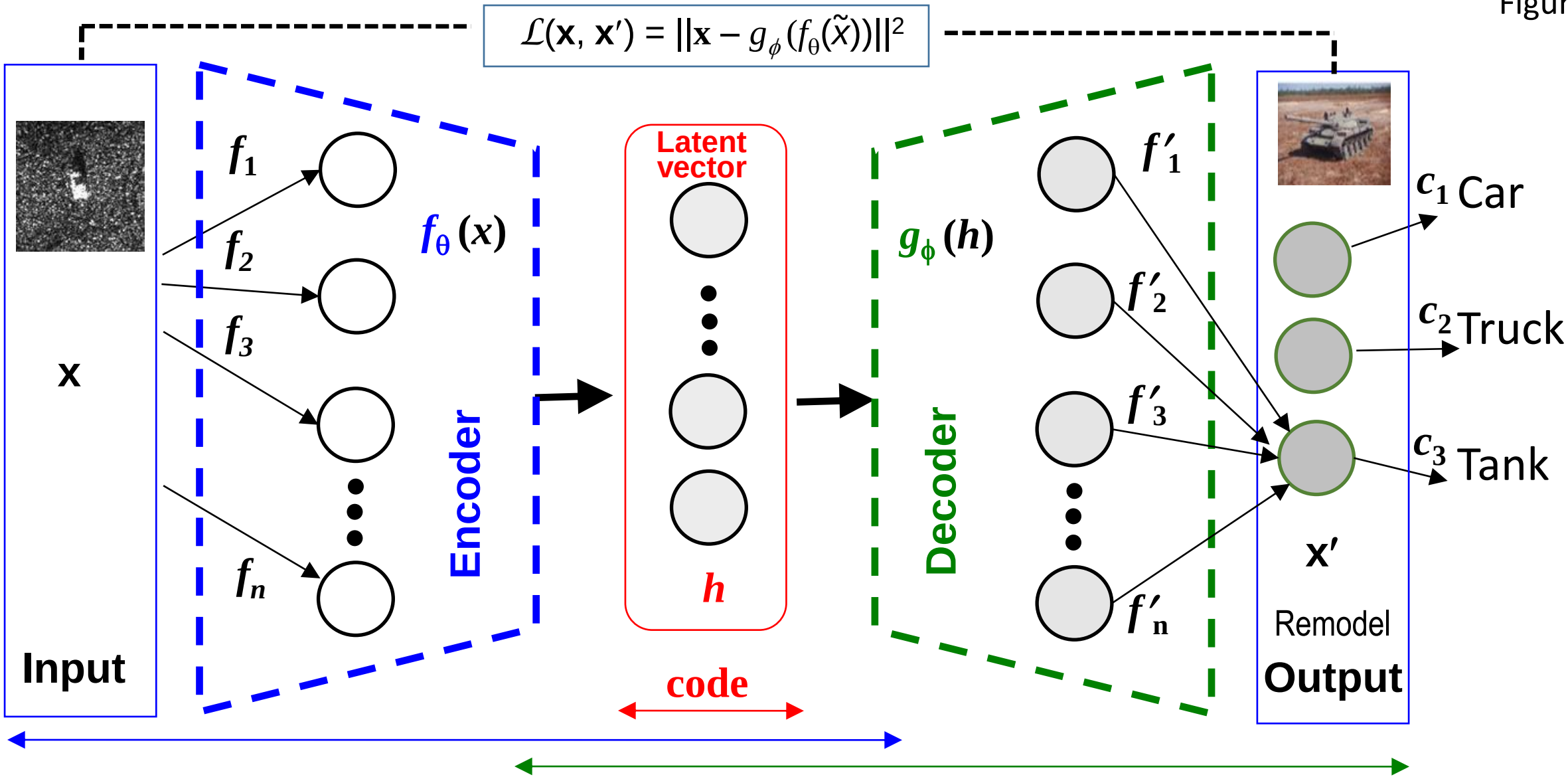
- Multi-Layer Perceptron
- Pulse-coupled NN (PCNN)
- **Deep Neural Network (DNN)**

- Deep stacking Nets (DSN)
- **Convolutional NN (CNN)**
- **Recurrent NN (RNN)**

Contemporary – Active learning, Transfer Learning
Generative Adversarial Networks (GANs)

Autoencoder (Unsupervised Learning)

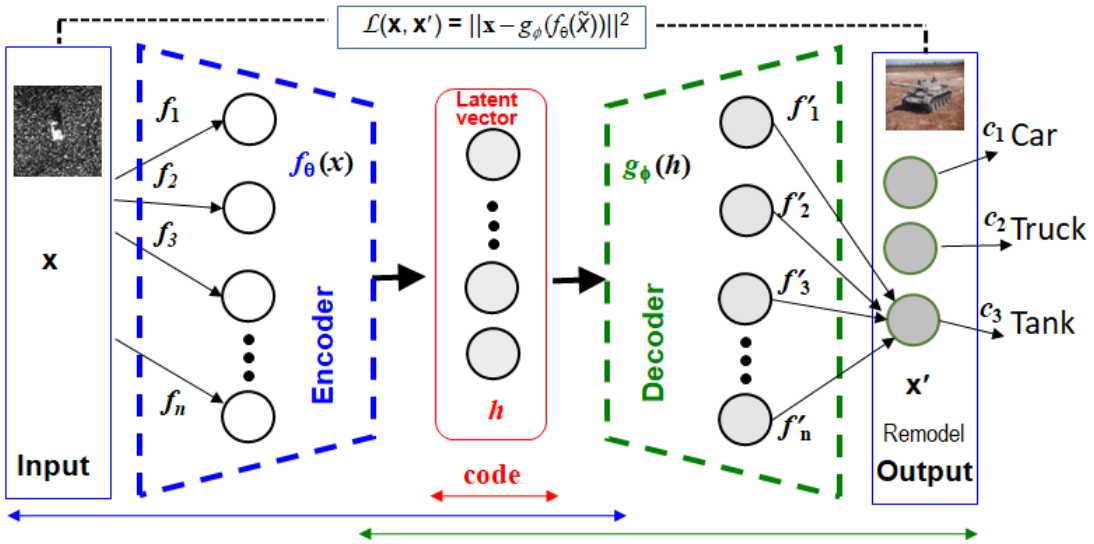
Figure 4.4



Autoencoder (Unsupervised Learning)

- An autoencoder is an **unsupervised learning** model that does not require labels to predict ***y*** = $f(\boldsymbol{x})$, but rather minimizes the difference between the input **x** and the output ***x′***.
- **Encoder**: mapping states to features (θ: X $\rightarrow$F).
- **Decoder**: mapping features to outputs (ϕ: F $\rightarrow$X).
- Encoder-decoder: **Minimize the error**:

$$\theta, \phi = \arg\min_{\theta,\phi} \left\| X - (\phi \bullet \theta) X \right\|^2$$



- Using a single hidden layer **h** (called the code or *latent* variables representation), the **encoder** maps the input **x** to the hidden layer (**W** is the weight and is the bias vector)
  - where σ is an element-wise activation function (e.g., rectified linear unit (ReLU), hyperbolic, or sigmoid function)

$$h = f_\theta(\tilde{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

sigmoid: $\sigma(x) = 1 / [1 + \exp(-x)]$

hyperbolic tangent (tanh) $\sigma(x) = [\exp(x) - \exp(-x)] / [\exp(x) + \exp(-x)]$

ReLU: $\sigma(x) = \max(x, 0)$.

- The **decoder** stage of the maps **h** to the reconstructed model **x ′**:

$$\mathbf{x}' = z = g_\phi(\mathbf{h}) = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

- The decoder minimizes the reconstruction errors from the loss function by averaging **x** over some input training set

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||^2$$

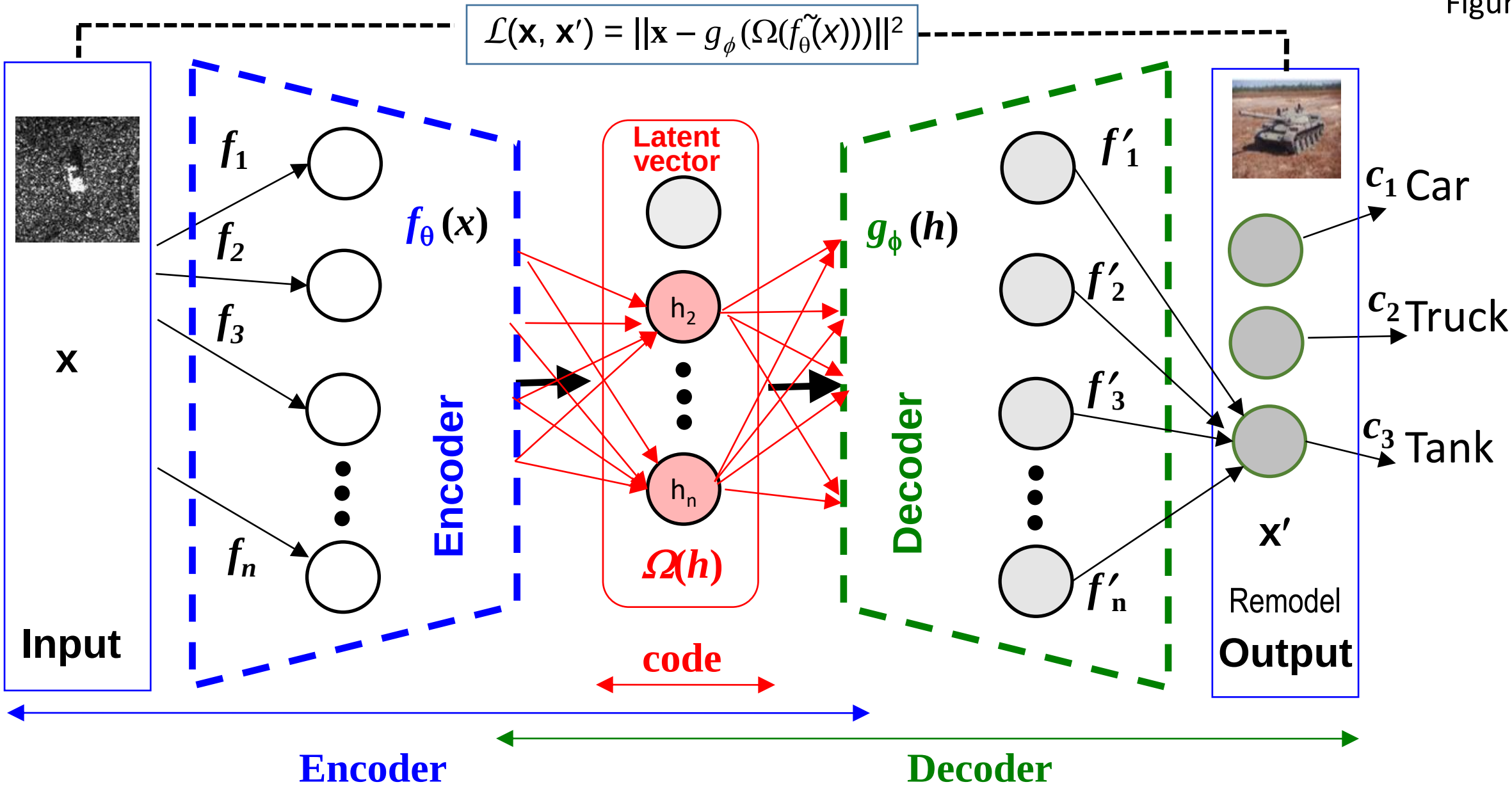
$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - g_\phi(f_\theta(\tilde{x}))||^2$$

- A **sparsity constraint** (or penalty $\Omega(\boldsymbol{h})$)

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \sigma'(\mathbf{W}'([\sigma(\Omega(\boldsymbol{h}))] + \mathbf{b}'||^2$$

Sparse Autoencoder

Figure 4.5



Autoencoder Method

- Some sparsity of **activation penalty terms** include:
- **Kullback-Leibler (KL) divergence (**average activation)

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}[h_j(x_i)] \qquad \Omega(\boldsymbol{h}) = \sum_{j=1}^{s} \mathrm{KL}(\rho||\hat{\rho}_j)$$

- **ℓ_1 or ℓ_2 regularization (scaling parameter λ)**

$$\Omega(\boldsymbol{h}) = \lambda\Sigma_{i=1} |h_i|$$

- **K-sparse autoencoder (**rectified linear units (ReLU) **)**

$$\hat{x}_i = \underset{z}{\operatorname{argmin}}\|x_i - \mathbf{W}\mathbf{x}\|_2^2 \qquad s.t. \quad \|\mathbf{x}\|_0 < k$$

- **Autoencoders:**
  - **Denoising AE (distortion**
  - **Contractive AE (Frobenius Norm)**
  - **Variational AE (Bayes update)**

ALGORITHM: Autoencoder
INPUT: penalty term $\Omega(\boldsymbol{h})$
INITIALIZE: $\mathbf{w} = (w_1, \ldots, w_n)$; $\mathbf{b} = (b_1, \ldots, b_n)$
For $t = 1, 2, \ldots, n$

- receive instance: $\mathbf{x}_t \in \mathbb{R}^n$
- encode: $\boldsymbol{h} = f_\theta(\tilde{\boldsymbol{x}}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$
- loss analysis: $\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - g_\phi(f_\theta(\tilde{\boldsymbol{x}}))\|^2 + \Omega(\boldsymbol{h})$
- by

1. determine $\Omega(\boldsymbol{h})$

$$\hat{\rho}_j = \frac{1}{m}\Sigma_{i=1}^{m}[h_j(x_i)] \qquad \text{(AE-KL)}$$

$$\Omega(\boldsymbol{h}) = \Sigma_{j=1}^{s}\left[\rho\log\left(\frac{\rho}{\hat{\rho}_j}\right) + (1-\rho)\log\left(\frac{1-\rho}{1-\hat{\rho}_j}\right)\right]$$

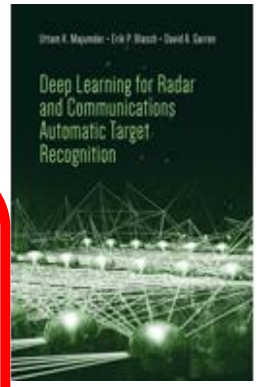
$$\Omega(\boldsymbol{h}) = \lambda\,\Sigma_{i=1}|h_i| \qquad \text{(AE-L-norm)}$$

2. Backprop error of k largest activations

$$\hat{x}_i = \operatorname{argmin}_z\|x_i - \mathbf{W}\mathbf{x}\|_2^2 \quad , \|\mathbf{x}\|_0 < k \qquad \text{(AE-KS)}$$

- decode update: $z = g_\phi(\mathbf{h}) = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}')$

Chapter 4 Outline

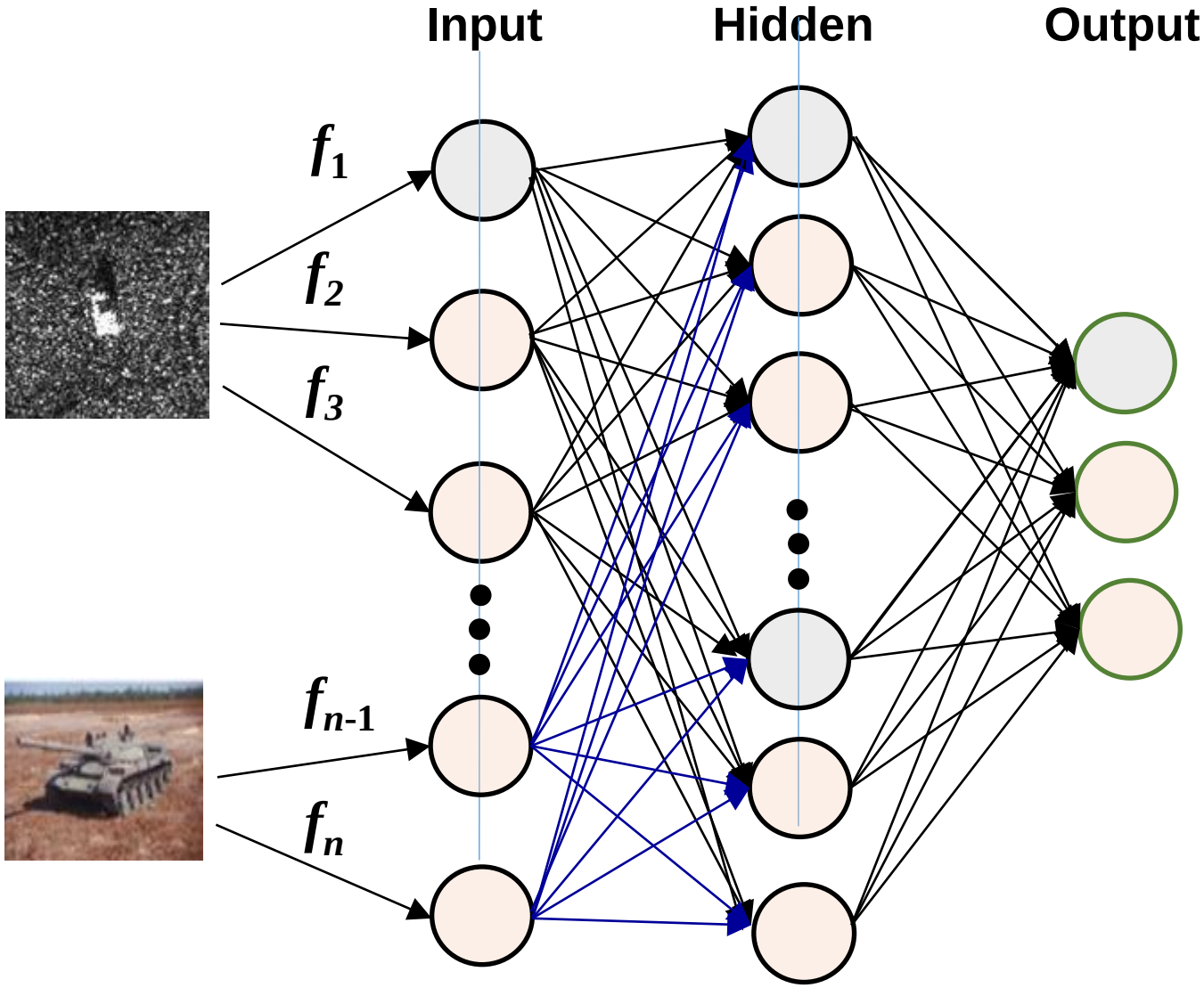


Feed Forward NN

- The methods are based on FNN
  - Multi-Layer Perceptron
  - Boltzmann NN
  - Pulse Couple NN

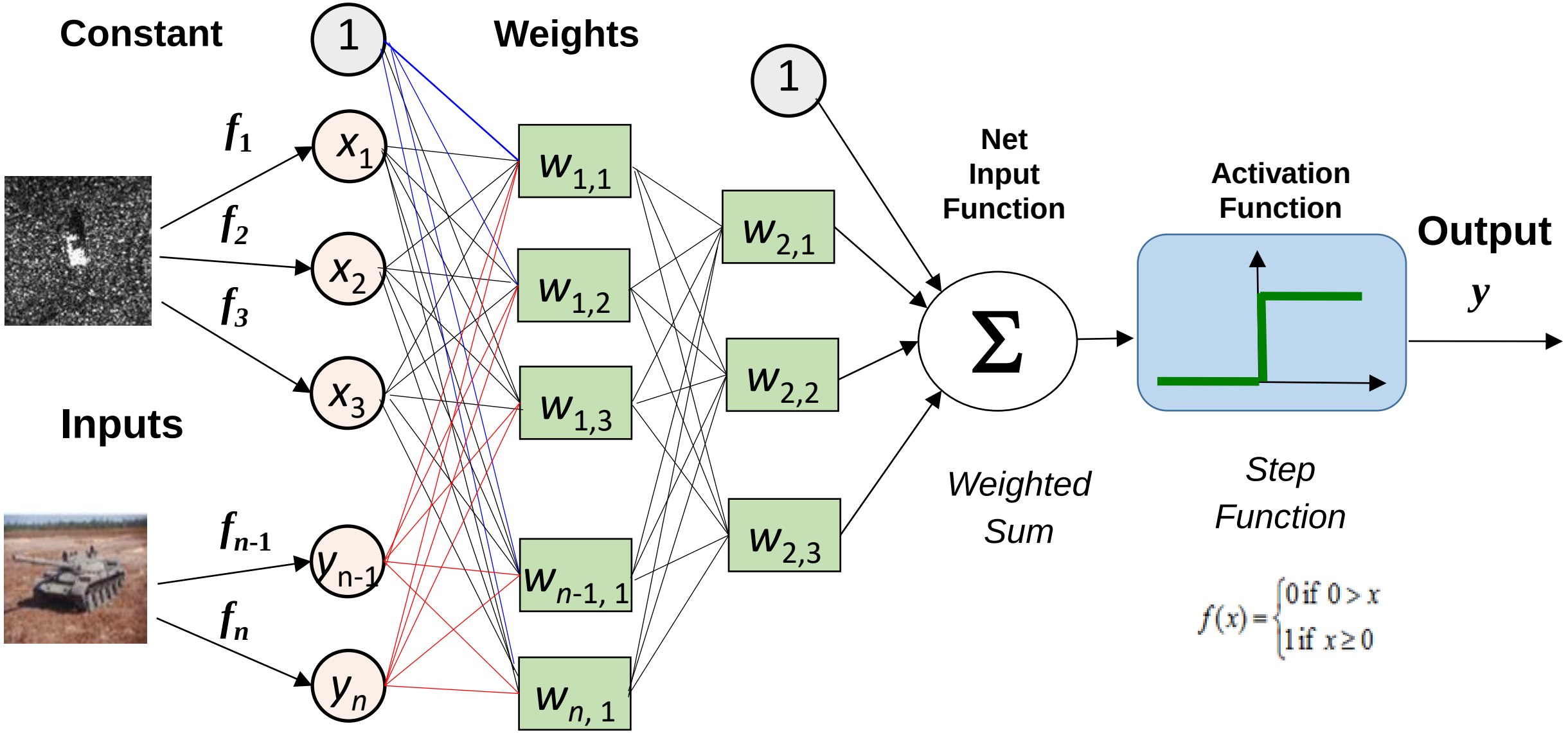
- New methods
  - Convolutional NN
  - Recurrent NN

Figure 4.6



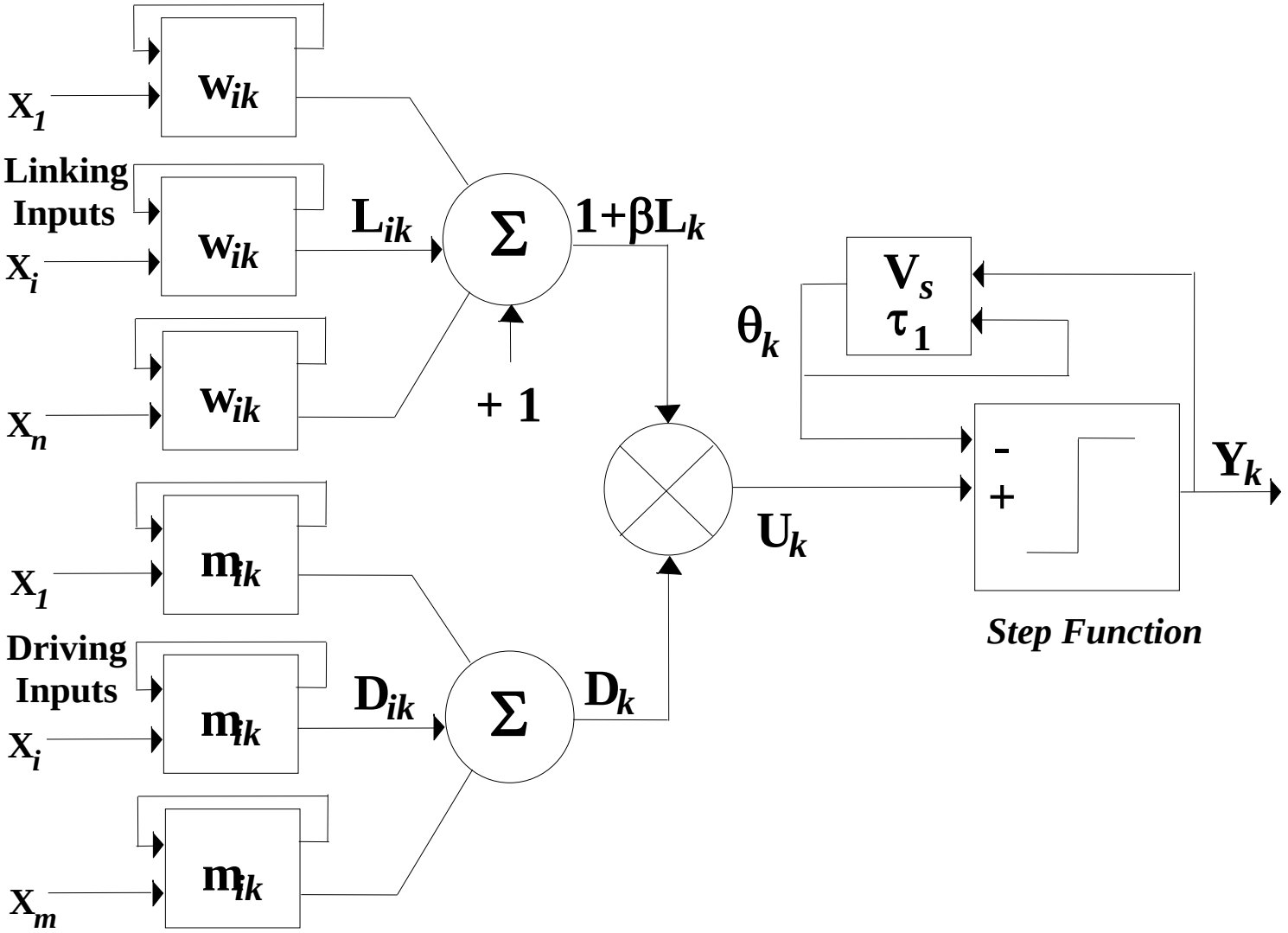
Multi-Layer Perceptron

Figure 4.7



Pulse Coupled NN

Figure 4.8



Driving Function

$$D_{ik}(n) = D_{ik}(n-1) \cdot e^{-1/\tau_D} + V^D \cdot X_i(n) \cdot w_{ik}^D$$

Linking Function

$$L_{ik}(n) = L_{ik}(n-1) \cdot e^{-1/\tau_L} + V^L \cdot Y_i(n) \cdot w_{ik}^L$$

Threshold Function

$$\theta_k(n) = \theta_o + \theta_k(n-1) \cdot e^{-1/\tau_s} + V^S \cdot Y_k(n)$$

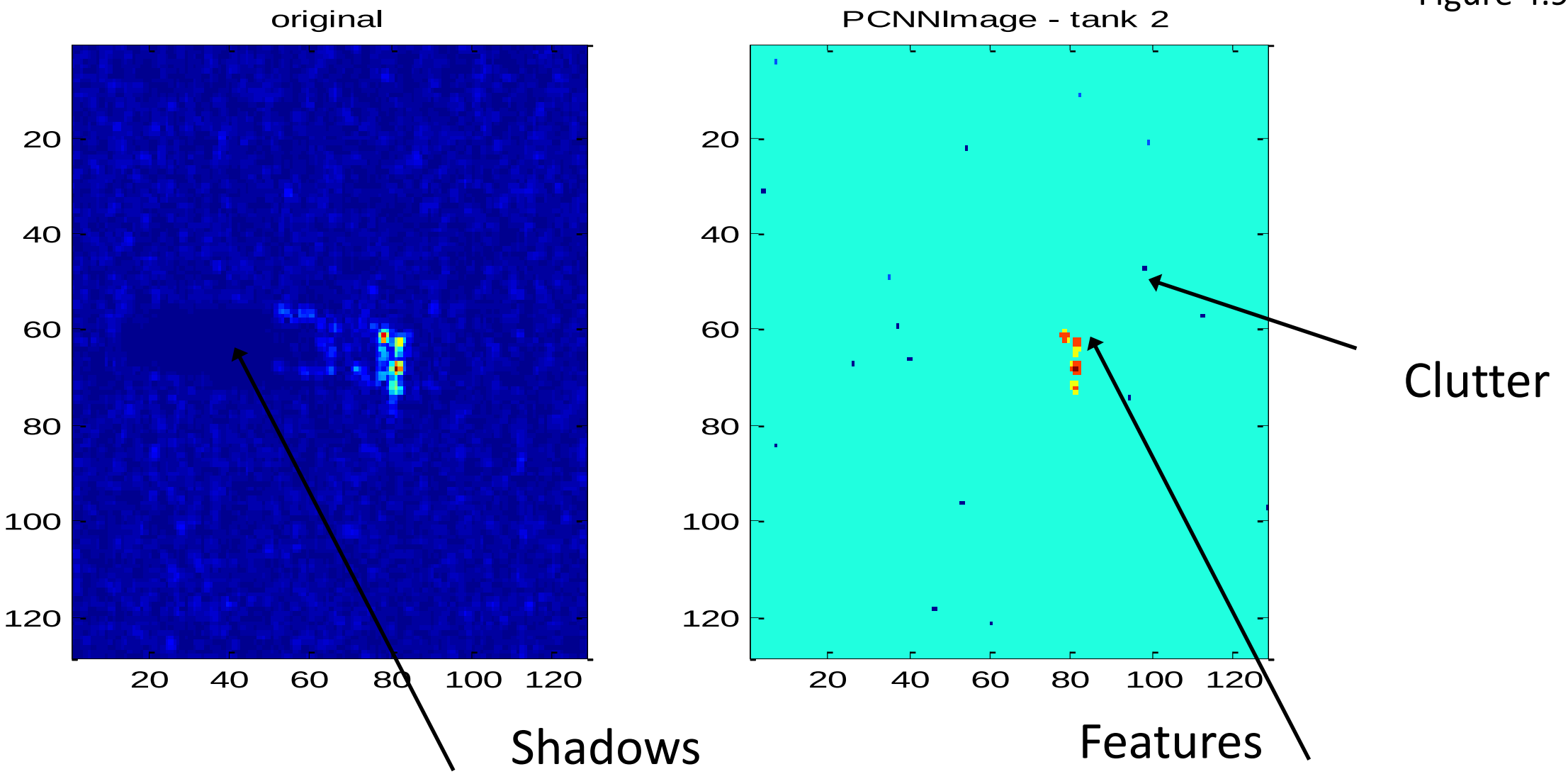
where

$$D_k = \sum D_{ik} \text{ and } L_k = \sum L_{ik}$$

Blasch, E. P., "Biological Information Fusion Using PCNN and Belief Filters," *IEEE Intl Joint Conference on Neural Networks,* vol.4, 2792 -2795, 1999.

PCNN on MSTAR Data

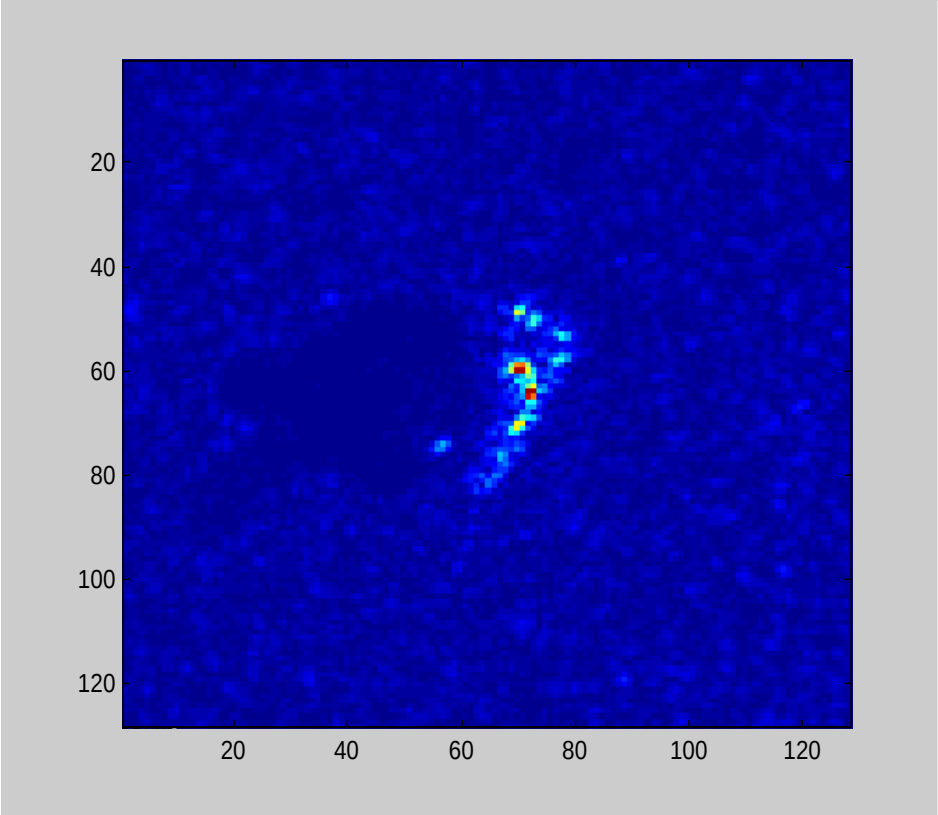
Figure 4.9



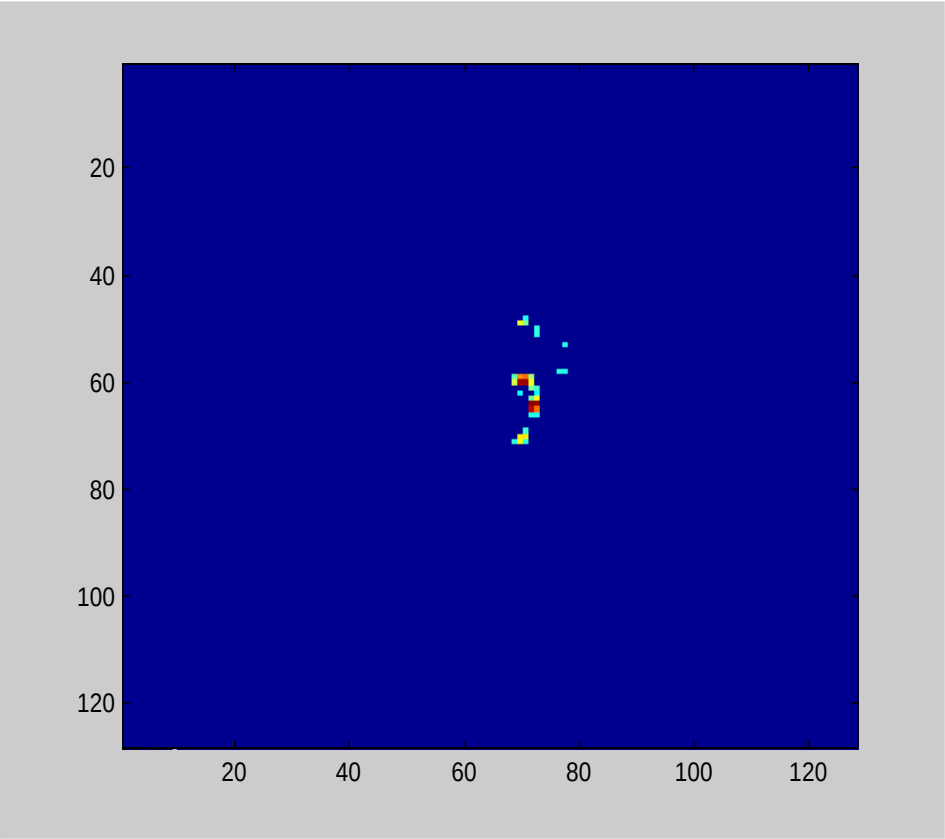
Blasch, E. P., "Biological Information Fusion Using PCNN and Belief Filters," *IEEE Intl Joint Conference on Neural Networks,* vol.4, 2792 -2795, 1999.

PCNN Results

Figure 4.10



(a) Target Image

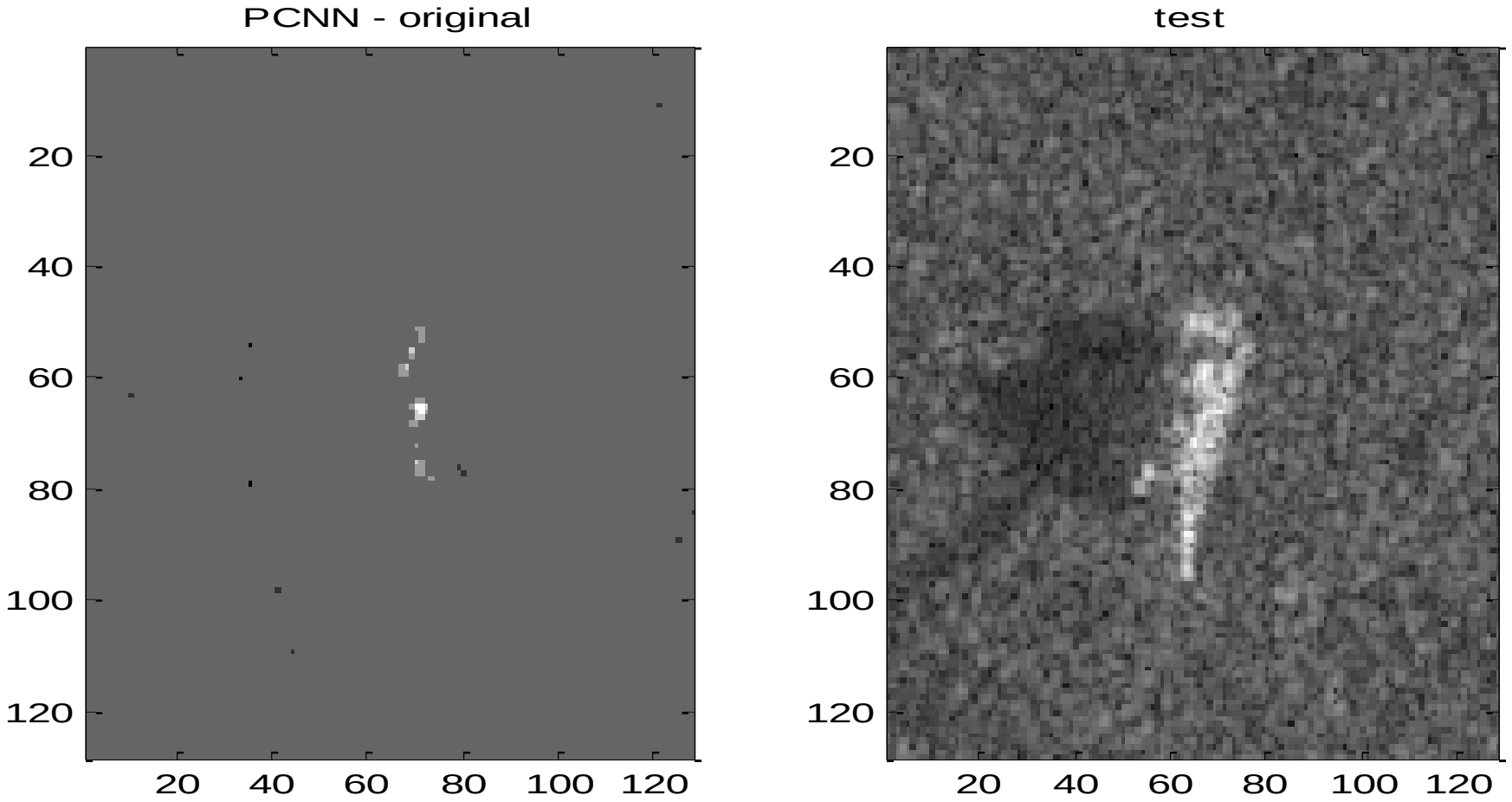


(b) PCNN Image

Blasch, E. P., "Biological Information Fusion Using PCNN and Belief Filters," *IEEE Intl Joint Conference on Neural Networks,* vol.4, 2792 -2795, 1999.

PCNN Results

Figure 4.11



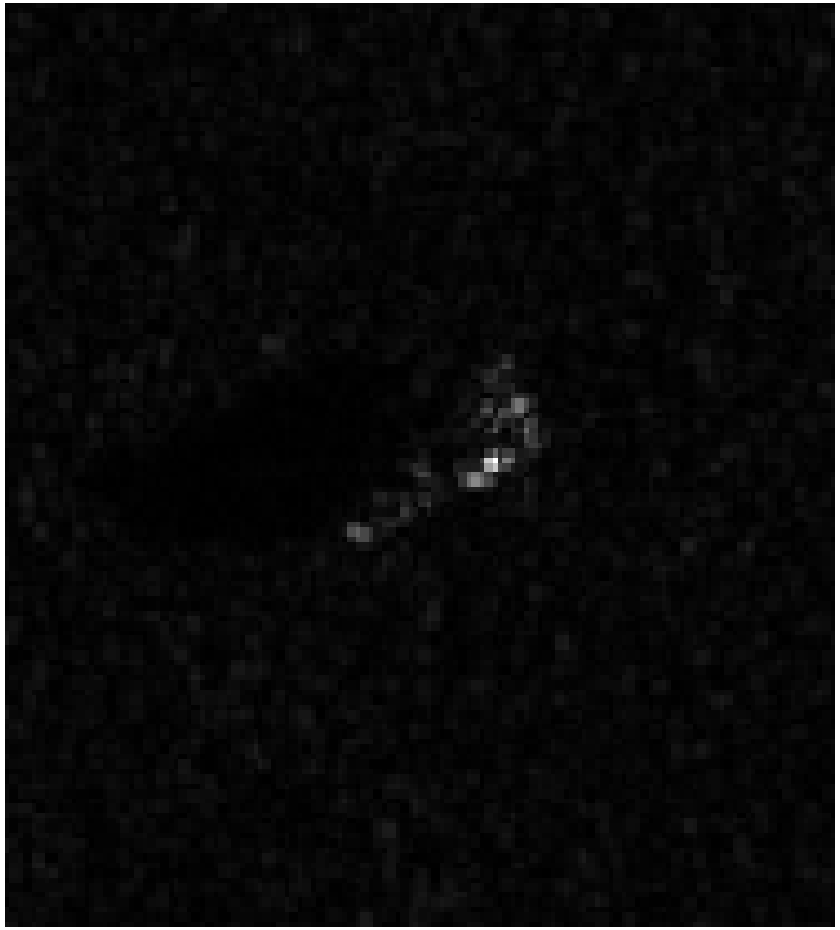
Blasch, E. P., "Biological Information Fusion Using PCNN and Belief Filters," *IEEE Intl Joint Conference on Neural Networks,* vol.4, 2792 -2795, 1999.

*360° Movie

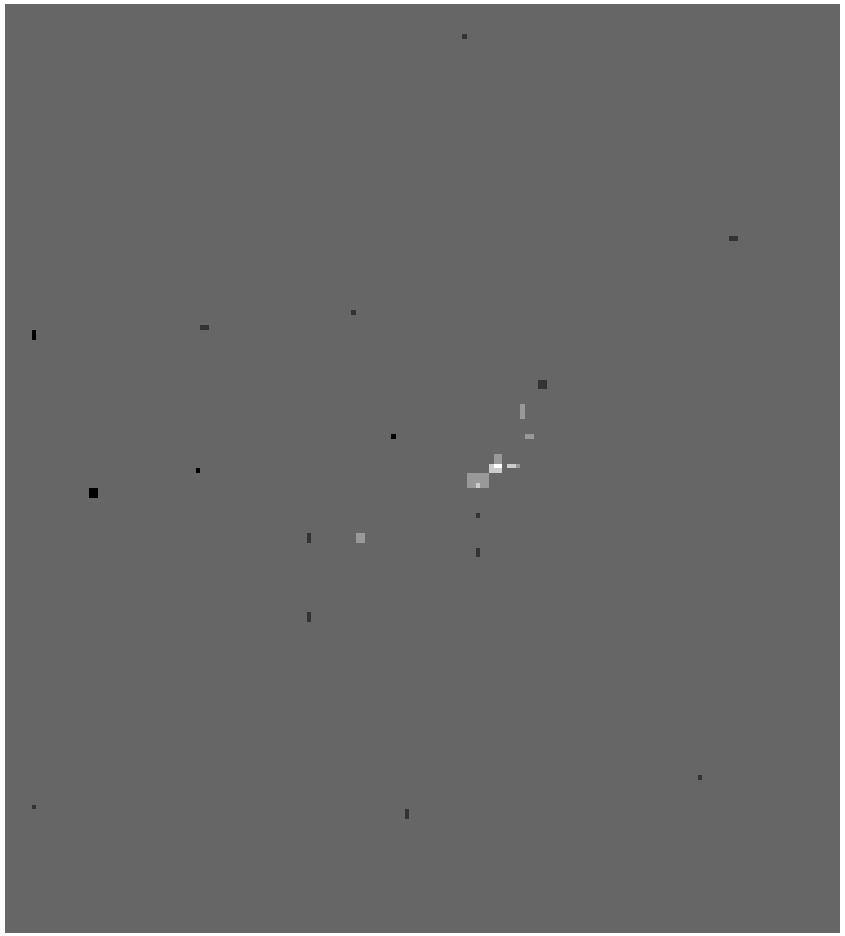
PCNN Results

Figure 4.12

original



PCNNImage - tank 2



Blasch, E. P., "Biological Information Fusion Using PCNN and Belief Filters," *IEEE Intl Joint Conference on Neural Networks,* vol.4, 2792 -2795, 1999.

*360° Movie

Confusion Matrix Fusion

Figure 4.13

MI ID SAR

	TARGET 1	TARGET 2	TARGET 3	TARGET 4	TARGET 5	TARGET 6	TARGET 7	TARGET 8	TARGET 9	TARGET 10	TARGET 11	OTHER
TARGET 1	0.66	0.04	0.03	0.03	0.02	0.03	0.03	0.03	0.03	0.03	0.04	0.03
TARGET 2	0.04	0.65	0.03	0.04	0.02	0.03	0.03	0.03	0.03	0.03	0.04	0.03
TARGET 3	0.04	0.04	0.65	0.04	0.02	0.03	0.03	0.03	0.03	0.03	0.04	0.03
TARGET 4	0.03	0.04	0.03	0.65	0.02	0.03	0.03	0.03	0.03	0.04	0.04	0.04
TARGET 5	0.03	0.03	0.03	0.03	0.68	0.03	0.04	0.03	0.03	0.03	0.03	0.03
TARGET 6	0.03	0.03	0.03	0.03	0.03	0.65	0.04	0.03	0.03	0.03	0.03	0.03
TARGET 7	0.04	0.04	0.04	0.04	0.04	0.05	0.55	0.04	0.04	0.04	0.04	0.04
TARGET 8	0.03	0.03	0.03	0.03	0.03	0.03	0.04	0.66	0.03	0.03	0.03	0.03
TARGET 9	0.03	0.04	0.03	0.04	0.02	0.03	0.03	0.03	0.63	0.04	0.04	0.04
TARGET 10	0.04	0.04	0.04	0.04	0.03	0.03	0.03	0.03	0.04	0.59	0.05	0.04
TARGET 11	0.04	0.04	0.04	0.04	0.03	0.03	0.03	0.03	0.04	0.04	0.61	0.04
NOT-IN-LIB	0.04	0.04	0.04	0.04	0.03	0.03	0.04	0.03	0.04	0.04	0.05	0.59

Single look

MI ID PCNN

	TARGET 1	TARGET 2	TARGET 3	TARGET 4	TARGET 5	TARGET 6	TARGET 7	TARGET 8	TARGET 9	TARGET 10	TARGET 11	OTHER
TARGET 1	0.69	0.1	0.05	0.02	0.02	0.02	0.02	0.04	0.02	0	0.02	0
TARGET 2	0.07	0.66	0.12	0.01	0.02	0.03	0	0.02	0.04	0	0.03	0.01
TARGET 3	0.04	0.13	0.63	0.03	0	0	0	0.02	0.06	0.01	0.04	0.03
TARGET 4	0.02	0.01	0.03	0.65	0	0	0	0	0.08	0.02	0.16	0.03
TARGET 5	0.02	0.04	0	0	0.73	0.16	0.05	0	0	0	0	0
TARGET 6	0.02	0.04	0	0	0.17	0.62	0.14	0	0	0	0	0
TARGET 7	0.04	0	0	0	0.1	0.26	0.6	0	0	-0	0	0
TARGET 8	0.05	0.04	0.05	0	0	0	0	0.82	0.03	0	0	0.01
TARGET 9	0.01	0.05	0.07	0.09	0	0	0	0.02	0.7	0	0.05	0.01
TARGET 10	0	0	0.04	0.04	0	0	-0	0	0	0.54	0.19	0.19
TARGET 11	0.01	0.02	0.02	0.08	0	0	0	0	0.02	0.04	0.75	0.05
NOT-IN-LIB	0	0.01	0.04	0.03	0	0	0	0.01	0.01	0.09	0.11	0.7

Multiple Looks

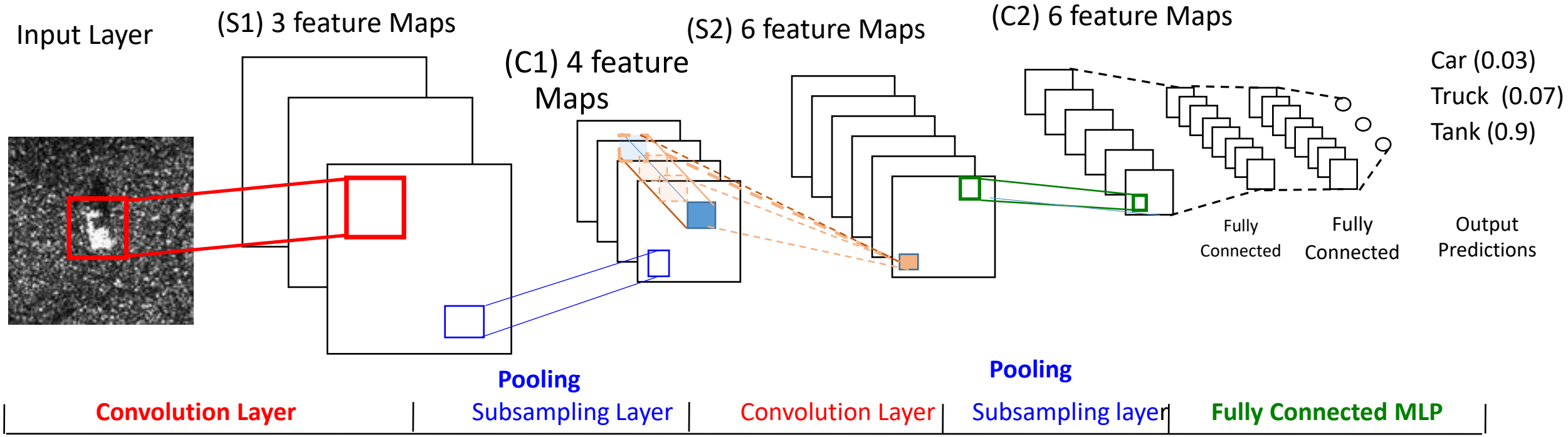
B. Kahler and E. Blasch, "Decision-Level Fusion Performance Improvement from Enhanced HRR Radar Clutter Suppression," *J. of. Advances in Information Fusion,* Vol. 6, No. 2, pp. 101-118, Dec. 2011.

* See Chapter 9 for more details

Convolutional Neural Network

- **Relevance (Spatial)**

Figure 4.14



Method	Convolutional Layers	Reference
AlexNet	5	Krizhevsky, *et al*. 2012
VGG (Visual Geometry Group)	19	Simonyan, *et al*., 2014
GoogleNet	22	Szegedy, *et al*., 2015
ResNet(Residual Network)	34	He, *et al*., 2015

CNN Processing

Figure 4.15

Three hyperparameters control the size of the output channel: the depth K (connections), stride S (spatial), and zero-padding P (output size). Number of neurons : N = [(T – K + 2P)/S + 1]

Image

0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Stride = 2

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

Convolution

0	1	-1	0
0	-1	-1	0
0	-3	0	0
0	0	0	0

0	-1	-1	0
0	-1	-3	0
0	-1	-2	0
0	0	0	0

Max Pooling

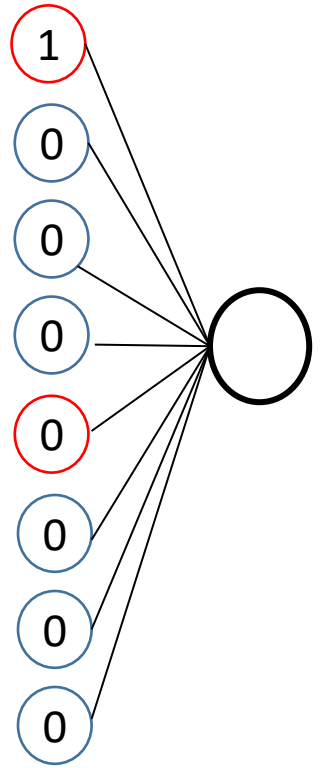
1	0
0	0

2 x 2 image

0	0
0	0

Each filter is a channel

Full Connect

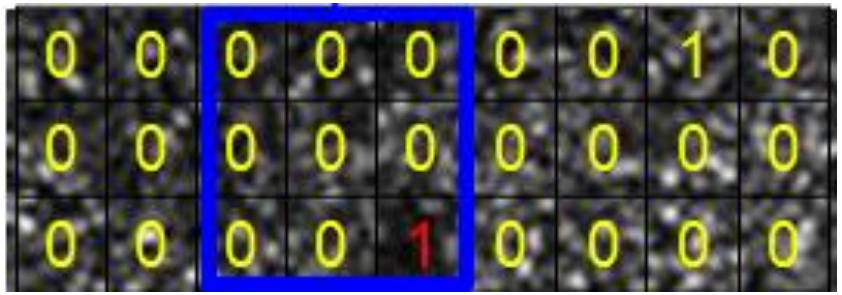


Softmax

Convolutional Neural Network

(1) Convolutional layer

A convolution layer convolves the result: $C = I * F$ where it is pointwise multiplication (row*column)



Image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Convolution

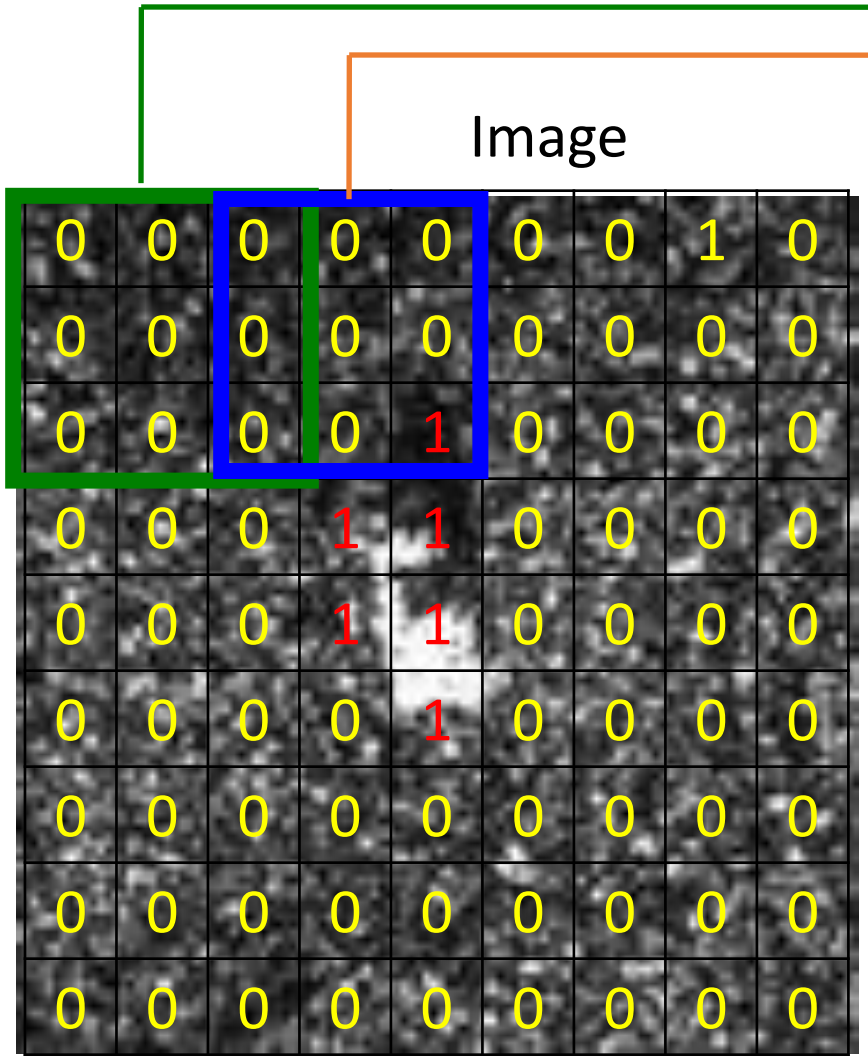
	1		

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$

= [0(1)+0(-1)+0(-1)] + [0(-1)+0(1)+0(-1)] + [0(-1)+0(-1)+(1)(1)] = 1

Convolutional Neural Network (2)

(1) Convolutional layer



Stride = 2

Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

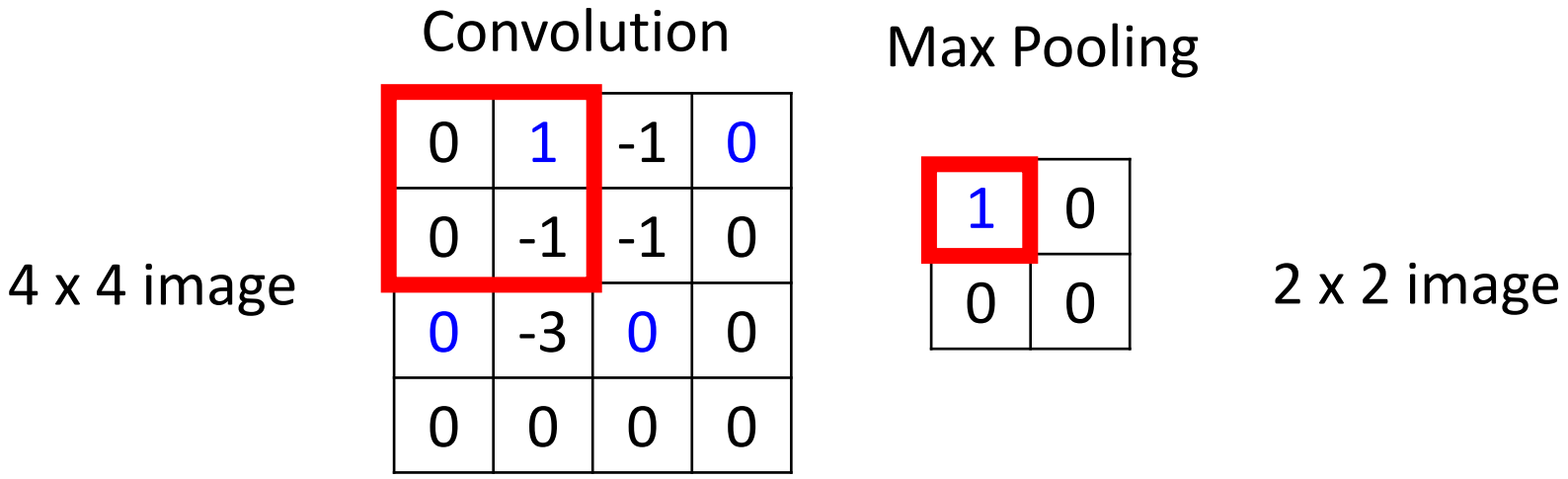
Convolution

0	1	-1	0
0	-1	-1	0
0	-3	0	0
0	0	0	0

0	-1	-1	0
0	-1	-3	0
0	-1	-2	0
0	0	0	0

Convolutional Neural Network
(2) Pooling layer

- The pooling layer **spatially resizes** the input and operates independently on every input depth. The most common filters size is 2×2 where a stride of $S = 2$ downsamples at every input depth slice in the input by 2 along both width and height, discarding 75% of the activations.
- Using the Rectified Linear Unit (ReLU), then the non-saturating activation function
  - $f(x) = \max(0, x)$ removes negative values from an activation map by setting them to zero.
  - The **max function** increases the nonlinear properties of the overall network decision without affecting the receptive fields of the convolution layer.
- Each filter the pooling layer goes from 4×4 to 2×2.



Convolutional Neural Network (3)

Convolution

0	1	-1	0
0	-1	-1	0
0	-3	0	0
0	0	0	0

0	-1	-1	0
0	-1	-3	0
0	-1	-2	0
0	0	0	0

Max Pooling

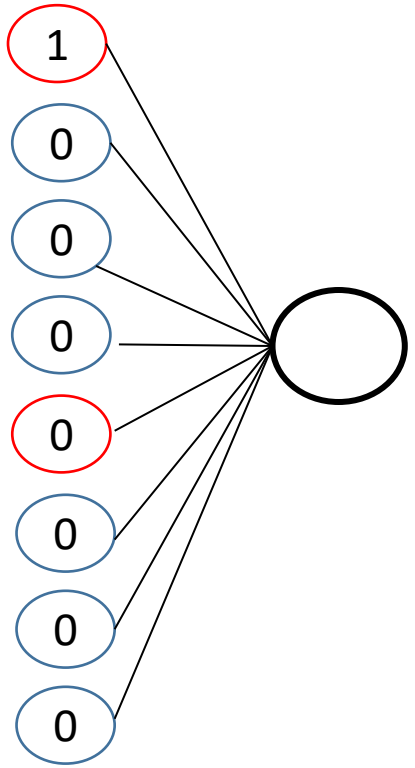
1	0
0	0

2 x 2 image

0	0
0	0

Each filter is a channel

Full Connect

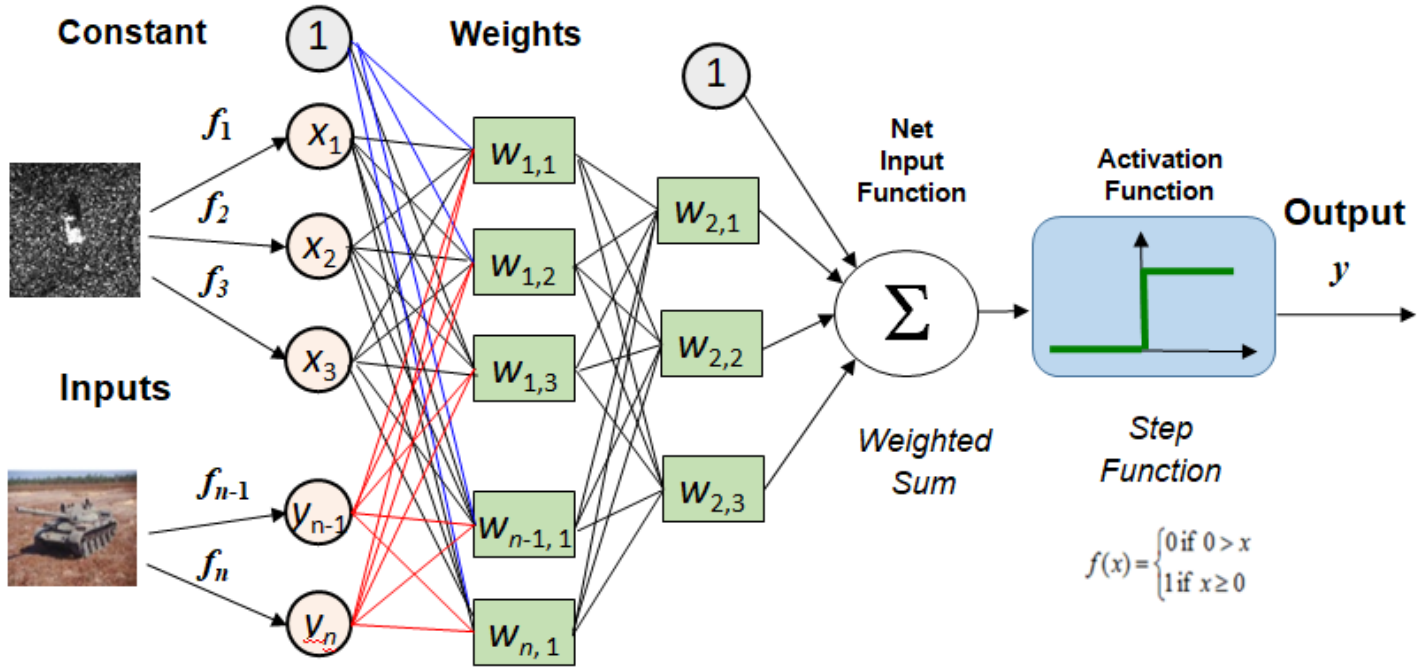


Softmax

Convolutional Neural Network

(3) Fully Connected Layer

- Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in the regular ANN.
- The **activations are given by a matrix multiplication** followed by a vector addition of a bias offset. The final layer is the '"loss layer," which penalizes the deviation between the predicted (output) and the true labels. Many CNNs use the *Softmax* loss to predict a single class from *K* mutually exclusive classes:



$$P(c|y_i) = \frac{\exp(z_c(y_i))}{\sum_{k \in [1,\ldots,K]} \exp(z_k(y_i))}$$

- where y_i is the output from the last hidden layer for corresponding target *c*, $z_c(\cdot)$ is the **final activation function**, and *k* is the one of target classes among the set [1, …, *K*].

Convolutional Neural Network

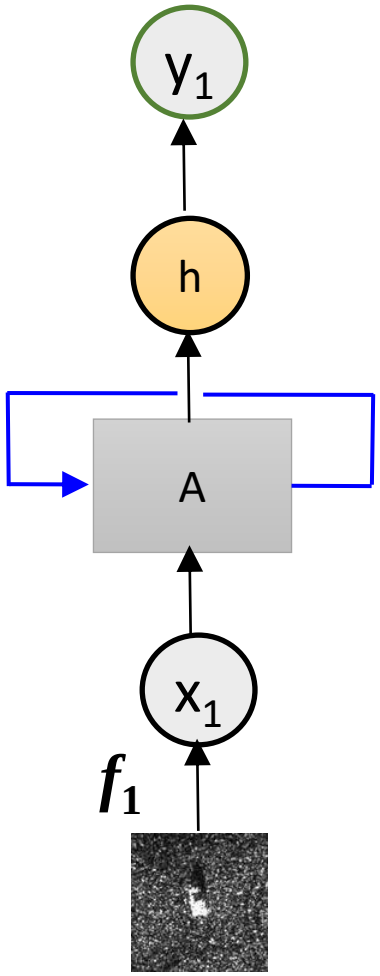
- **Drop Out:**
  - To **prevent overfitting**, at each training stage, individual nodes are either "dropped out" of the net with probability $(1 - p)$ or kept with probability p , **reducing the network size**.
  - In the training stages, the probability that a hidden node will be dropped is usually 0.5.
  - The removed nodes are then reinserted into the network with their original weights for testing.
  - DropOut Layer **improves training speed** by reducing node interactions, allowing them to learn more robust features that better generalize to new data.

Method	Performance (%)			Reference
	SOC	EOC-1	EOC-2	
Pulse Couple Neural Network (PCNN)	87.0	75	67	Blasch, 1998
Conditional False Alarm Rate (CFAR)	89.0	91	85	Mossing, *et al.*, 1998
Support Vector Machine (SVM)	90.0	81	75	Zhao, *et al.*, 2001
Conditional Gaussian	92.0	80	79	O'Sullivan, *et al.*, 2001
AdaBoost	92.0	82	78	Sun, *et al.*, 2007
Bayesian compressive sensing (BCS)	92.0	-	-	Zhang, *et al.*, 2013
Sparse Representation of Monogenic Signal (MSRC)	93.6	88.4	-	Dong, *et al.* 2014
Iterative graph thickening (IGT)	95.0	85	80	Srinivas, *et al.*, 2014
Modified Polar Mapping Classifier (M-PMC)	98.8	-	97.3	Park, *et al.*, 2014
Monogenic scale space (MSS)	96.6	98.2	-	Dong, *et al.*, 2015
A-ConvNets	99.1	96.1	98.9	Chen, *et al.*, 2016
AFRLeNet	99.4			Profeta *et al.*, 2016

Profeta *et al.* achieved a Portability of Correct Classification (PCC) of 99.4% for the SOC for a 7- target MSTAR set. using a *dropout layer* to AlexNet, which they called AFRLeNet

Recurrent Neural Network

Figure 4.16



RNN Loop

- **Persistence (Temporal)**

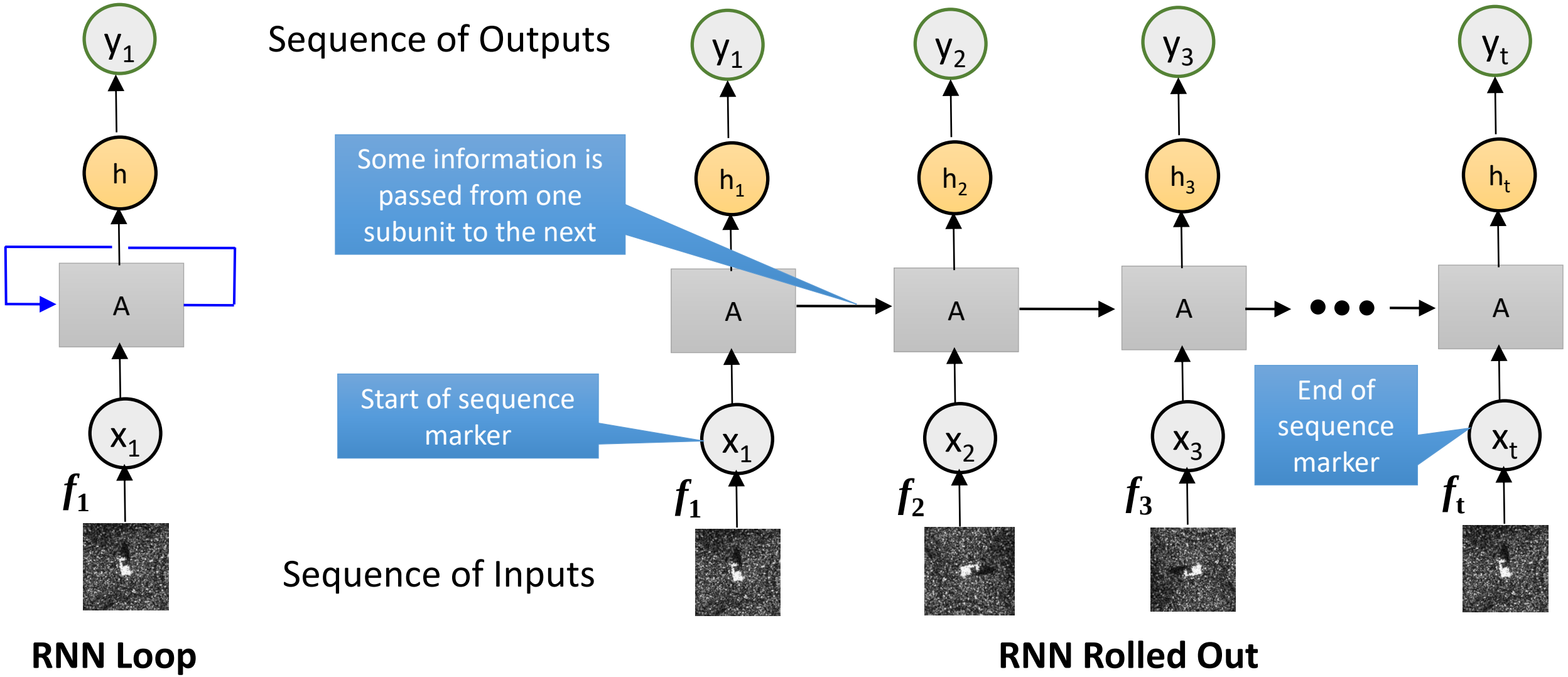
- RNNs maintains information persistence via loops, as with Gaussian Mixture Models (GMM).
  - A given neural network *A* receives an input x_t and outputs a value y_t, according to a hidden (latent relationship) h_t.
  - By passing data through iterative steps in the network, the **loop is realized**, and hence is recurrent.

- **Approaches**
  - Boltzmann Machine – NN that receives stochastic inputs
  - Long Short-term Memory (LSTM) – locally-stored short-term results
  - Gated Recurrent Unit (GRU) – efficient LSTM/RNN
  - Reservoir Computing - (e.g., Liquid State Machine) – for low power applications
  - Spiking Neural Network - randomly connected neurons

Recurrent Neural Network (2)

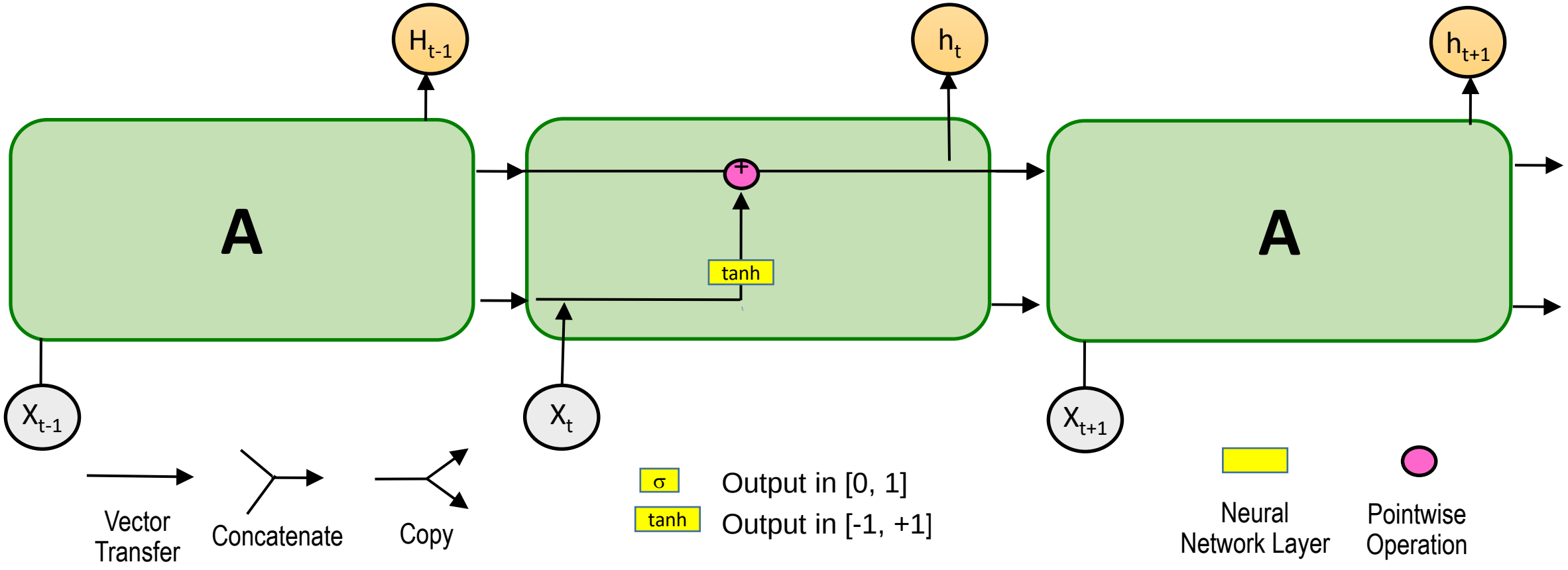
(a) RNN Loop, (b) RNN rolled out

Figure 4.16



Recurrent Neural Network

Figure 4.17



LSTM, there are **three functions**: (1) vector processing, (2) NN layer operations, and (3) pointwise operations

Vector operations include **concatenation** (merging) and **copying** (forking) of entire vector from one node to the next.

NN layers conduct the learning operations such as **sigmoid** (σ) and **tanh** weighting.

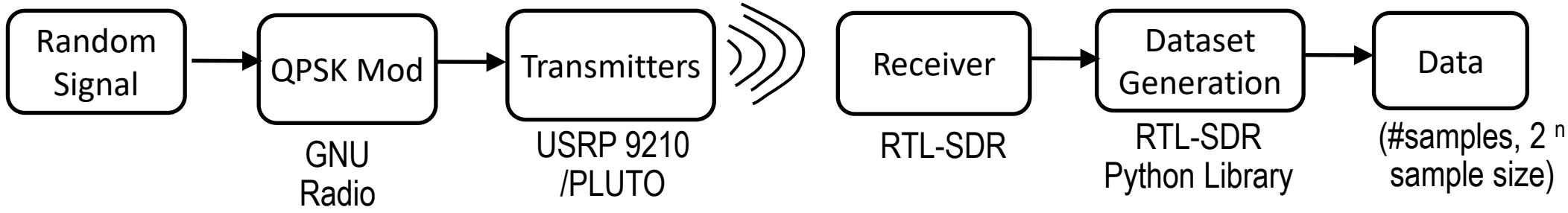
Pointwise operations include vector **addition**(+) and **multiplication** ($\times$).

RF Signal Data Collection

Figure 4.18

- Example
  - *Radio Frequency Adversarial Learning (RFAL)* framework exploits transmitter specific "signatures" like the **in-phase (I) and quadrature (Q) imbalance** (i.e., the I/Q imbalance) present in all transmitters
  - Learns feature representations using a deep neural network (DNN) that uses the I/Q data from received signals.
  - After elimination of the adversarial transmitters, the trusted transmitters are classified using a convolutional neural network (CNN), a fully connected DNN, and a recurrent neural network (RNN).

Signal Processing and Data Collection

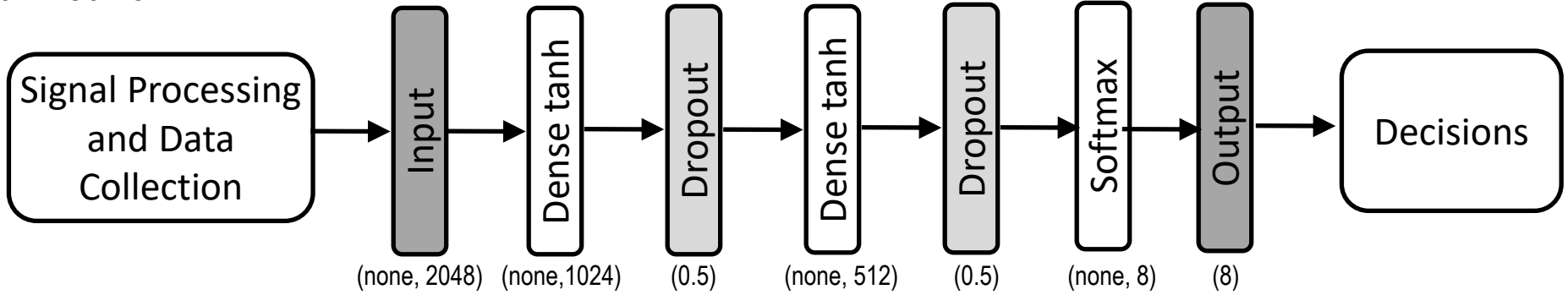


	# Transmitters	Type	Rows	Columns	SNR	Size
Test 1	4	USRP B210 transmitters	160K	2048	30 dB	6.8 GB
Test 1	8	USRP B210 transmitters	320K	2048	30 dB	13.45 GB
Test 2	1	USRP B210 transmitter	80K	2048	30 dB	3.31 GB
Test 2	1	PLUTO transmitter	80K	2048	30 dB	2.85 GB
Test 3	8	USRP B210 transmitters	320K	2048	20, 10, 0 dB	13 GB

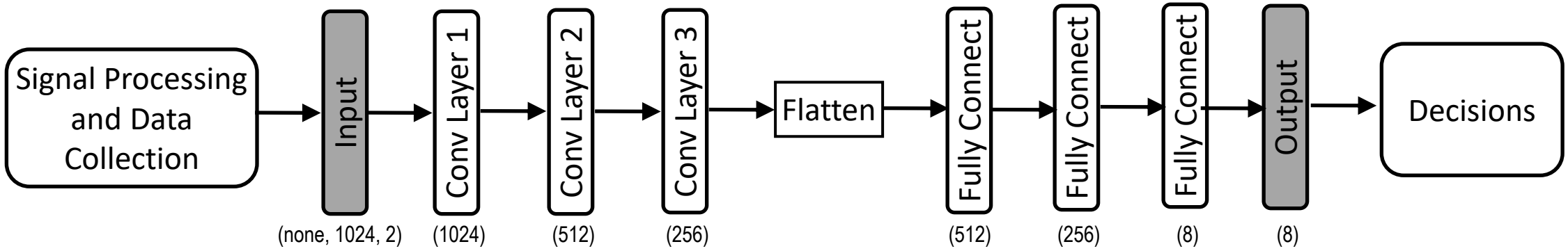
RF Deep Learning Architectures

Figure 4.19

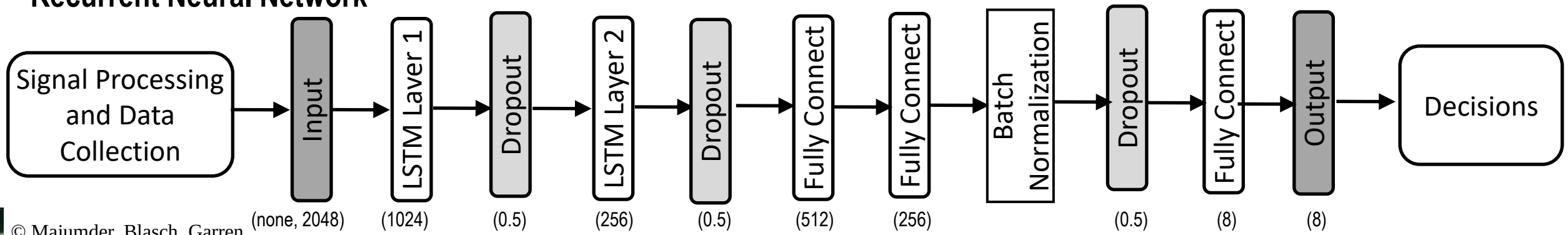
Deep Neural Network



Convolutional Neural Network



Recurrent Neural Network



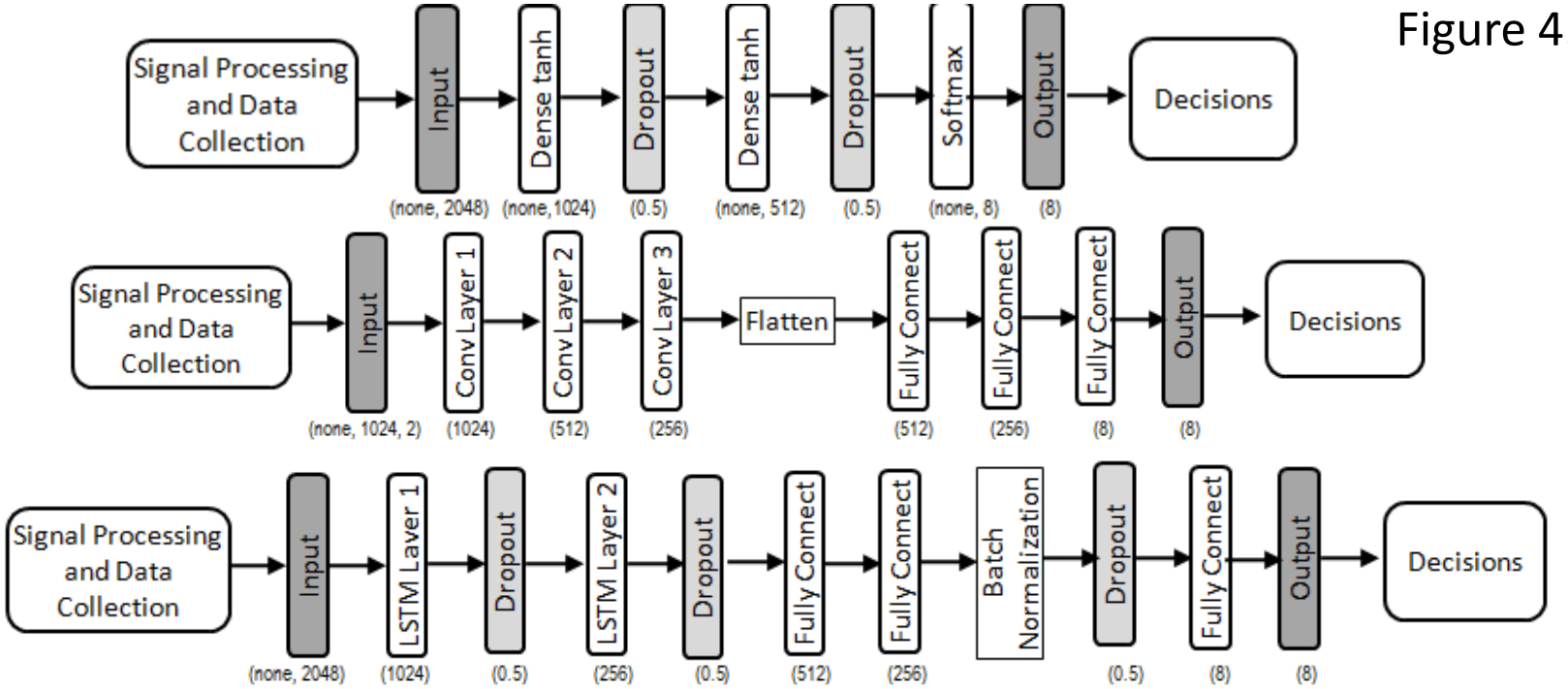
RF Deep Learning Architectures

Figure 4.19

Deep Neural Network

Convolutional Neural Network

Recurrent Neural Network

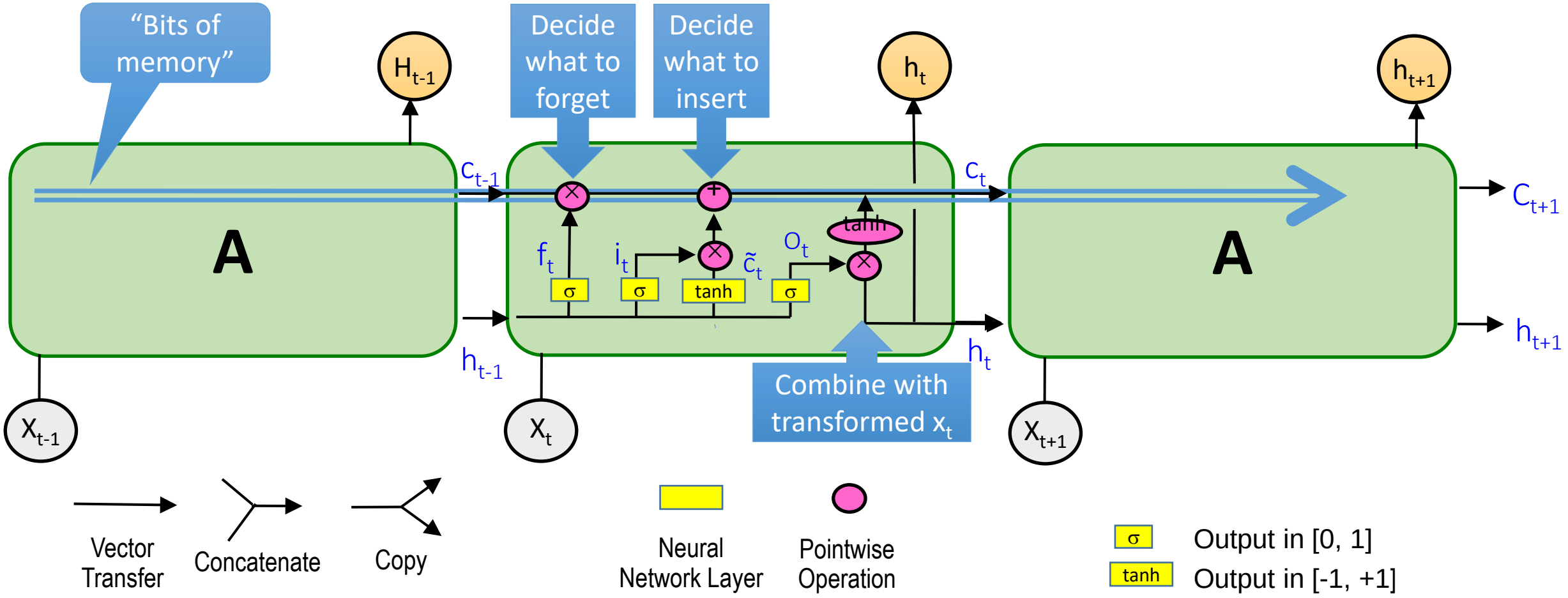


Accuracy Comparison of NN Approaches to classify 4 and 8 transmitters

#Trans	Model	#Parameters	Acc (%)	Time (min)
4	CNN	38 Million	89.07	~ 25
4	DNN	6.8 Million	96.49	~ 12
4	RNN-LSTM	14.2 Million	**97.40**	~ 12
4	RNN-GRU	10.7 Million	**97.85**	~ 12
8	CNN	38 Million	81.59	~ 30
8	DNN	6.8 Million	94.60	~ 15
8	RNN-LSTM	14.2 Million	**95.78**	~ 16
8	RNN-GRU	10.7 Million	**97.06**	~ 16

Long Short-Term Memory

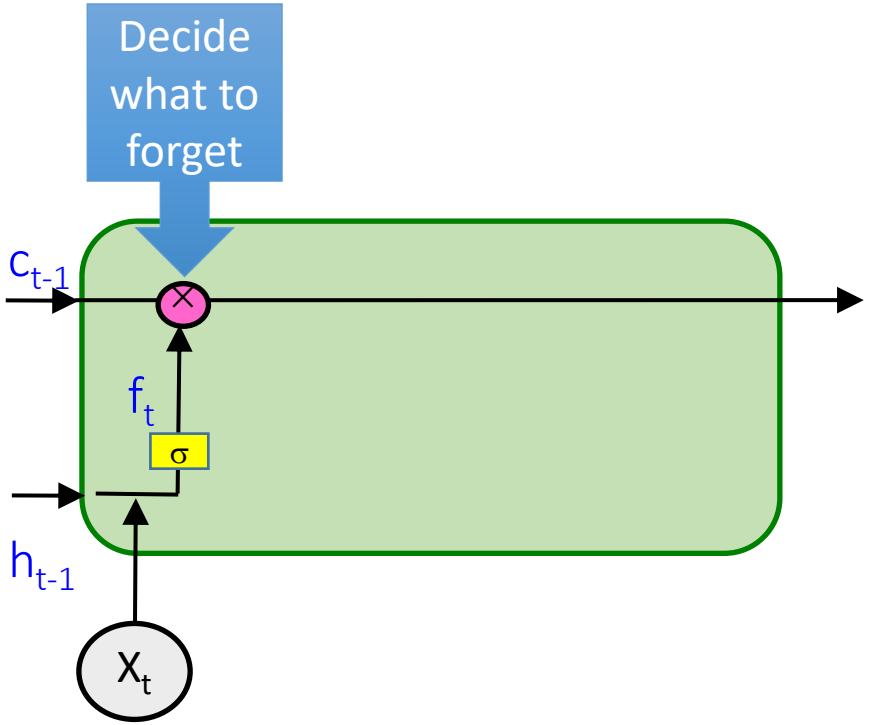
Figure 4.20



Long Short-Term Memory

Step 1: **Forget Gate** Layer

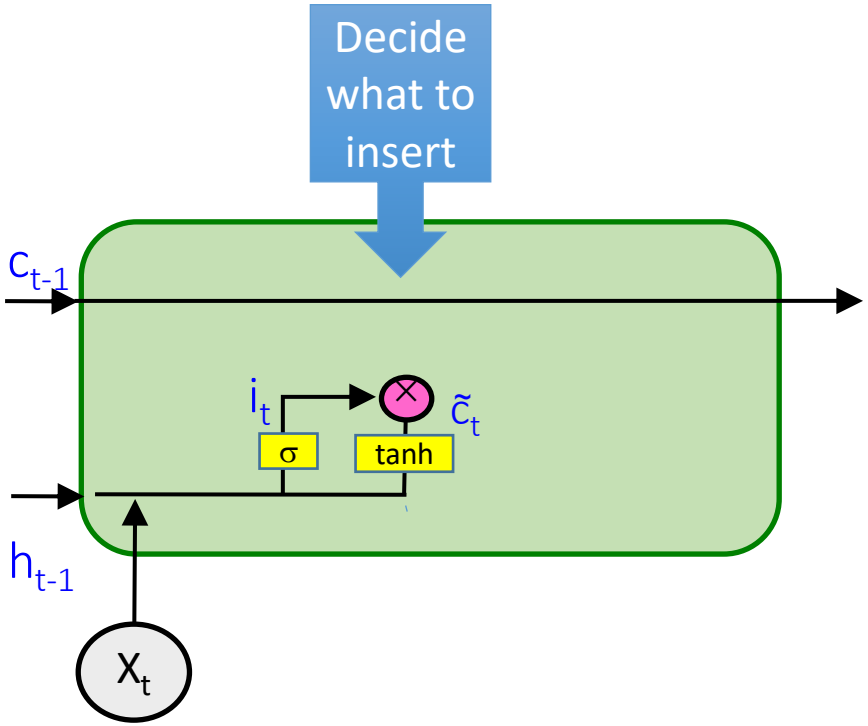
(**Data to remove**)



Features $f_t = \sigma\,(W_f \bullet [h_{t-1}, x_t] + b_f)$

Step 2: **Update Gate**

(**Data to store**)



Candidate (features)

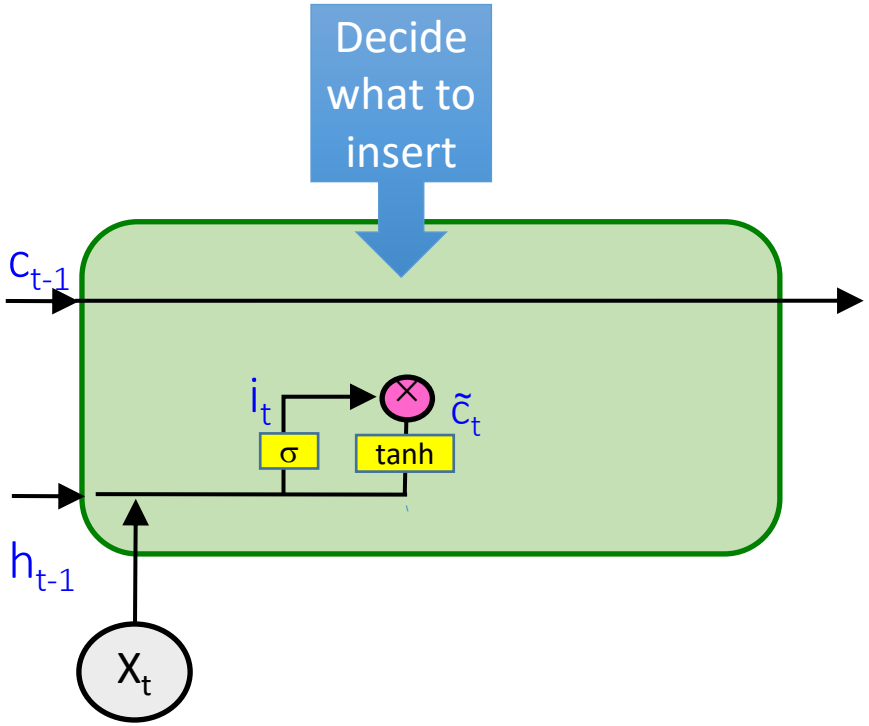
$$i_t = \sigma\,(W_i \bullet [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh\,(W_C \bullet [h_{t-1}, x_t] + b_C)$$

Long Short-Term Memory

Step 2: Update Gate

(**Data to store**)

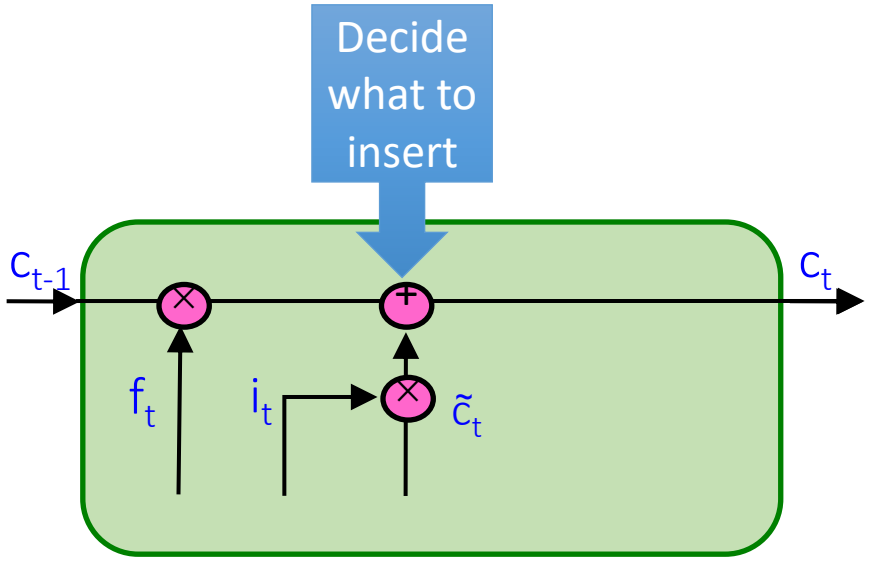


Candidate (features)

$$i_t = \sigma\,(W_i \bullet [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh\,(W_C \bullet [h_{t-1}, x_t] + b_C)$$

The old state f_t is multiplied, thereby forgetting information that is deemed to be unimportant.



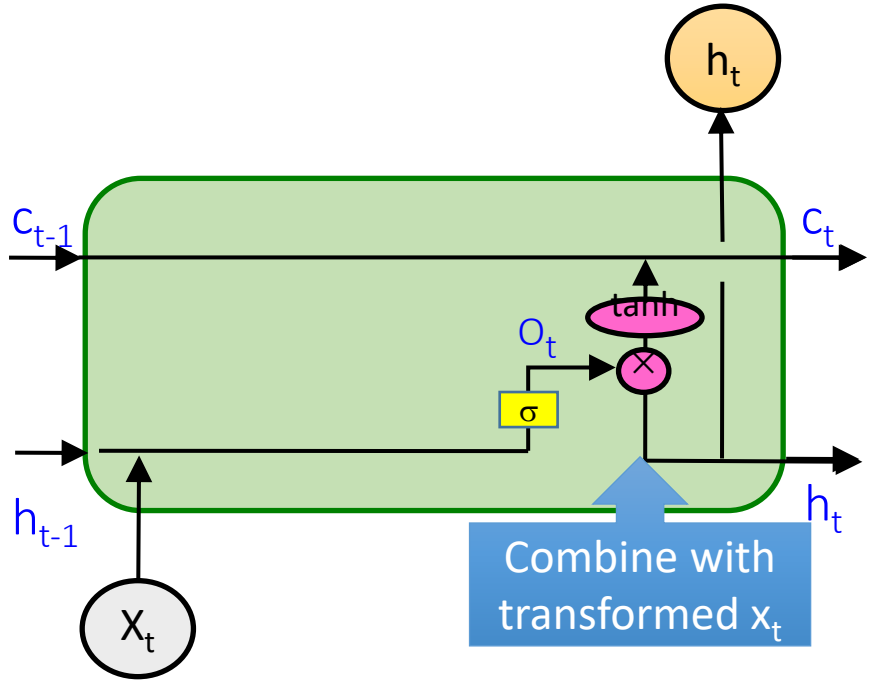
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step 1: $f_t = \sigma\,(W_f \bullet [h_{t-1}, x_t] + b_f)$

Long Short-Term Memory

Step 3: **Output Gate** (Data to Pass)

The filtered output includes a sigmoid layer that decides “what” and a tanh layer (e.g., {-1 +1}) to determine “how much”.

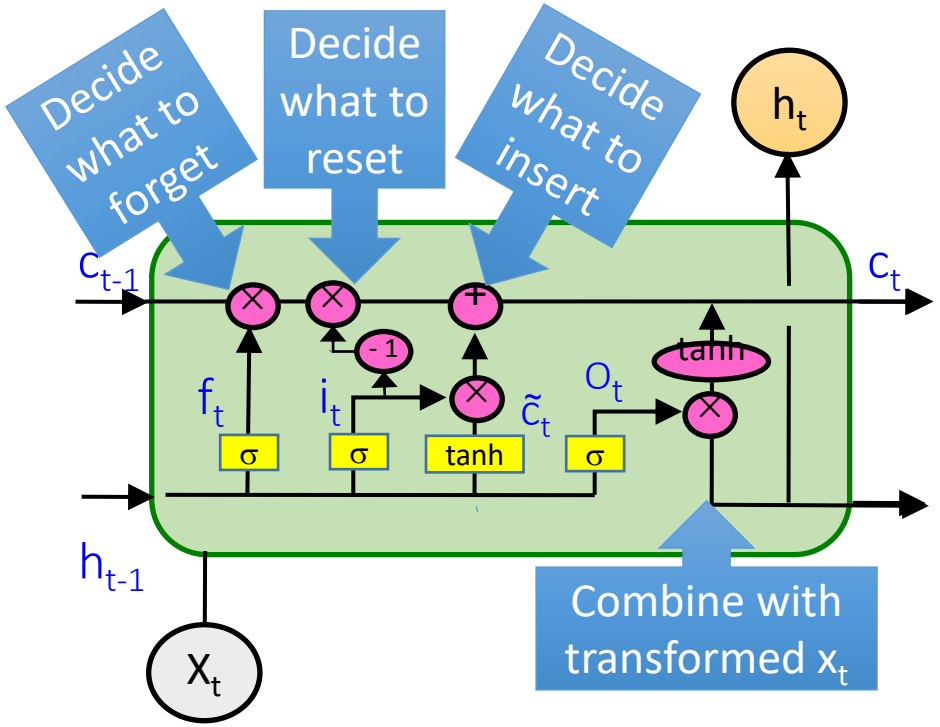


$$o_t = \sigma\,(W_o \bullet [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Gated Recurrent Unit (GRU)

One Sequence:



LSTM cell drawback is the need for additional memory

GRU have one less gate than LSTMs

- Combines the "forget" and "input" gates into a single "update gate."
- Simply exposes the full hidden content without any control.

"**reset gate**" (i_t), $i_t = \sigma\,(W_{xi}\,x_t + W_{hi}\,h_{t-1} + b_i)$

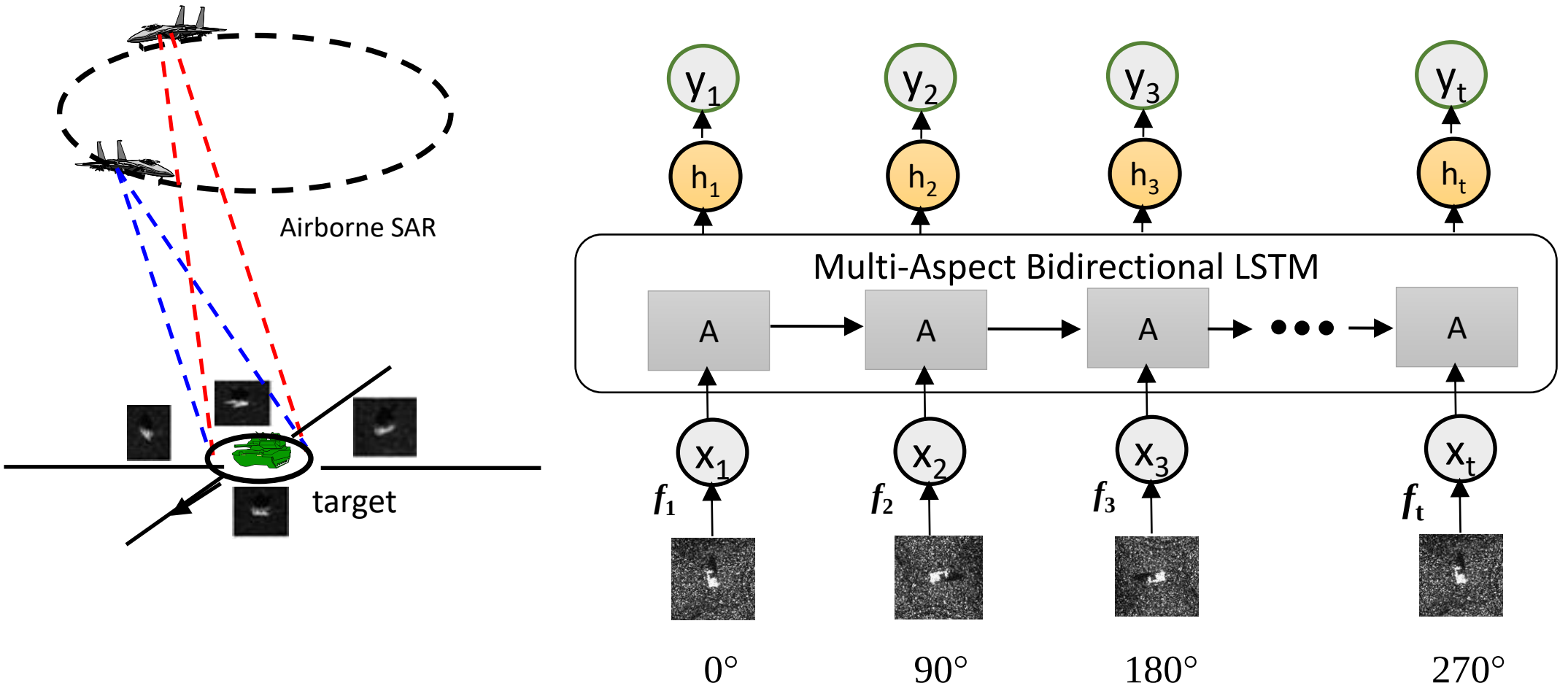
"**update gate**" (f_t), $f_t = \sigma\,(W_{xf}\,x_t + W_{hf}\,h_{t-1} + b_f)$

"**cell state**" memory (c_t) $C_t = \tanh\,(W_{xc}\,x_t + W_{hc}(f_t \circ h_{t-1}))$

"**recurrent**" (c_t) $h_t = (1 - i_t) \circ C_t + (i_t) \circ h_{t-1}$

LSTM SAR Example

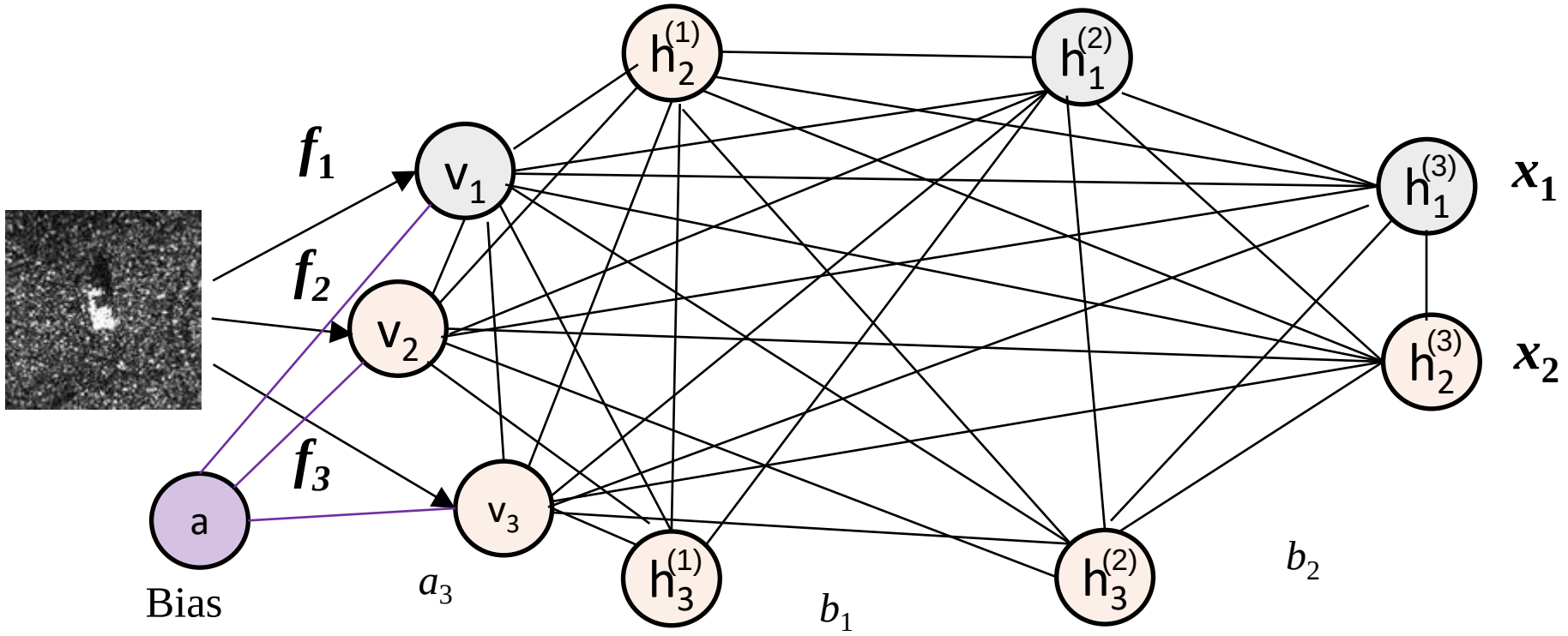
Figure 4.21



Multi-aspect aware Bidirectional Long Short-Term Memory (LSTM) recurrent neural networks (MA-BLSTM).

Boltzmann Machine

Figure 4.22



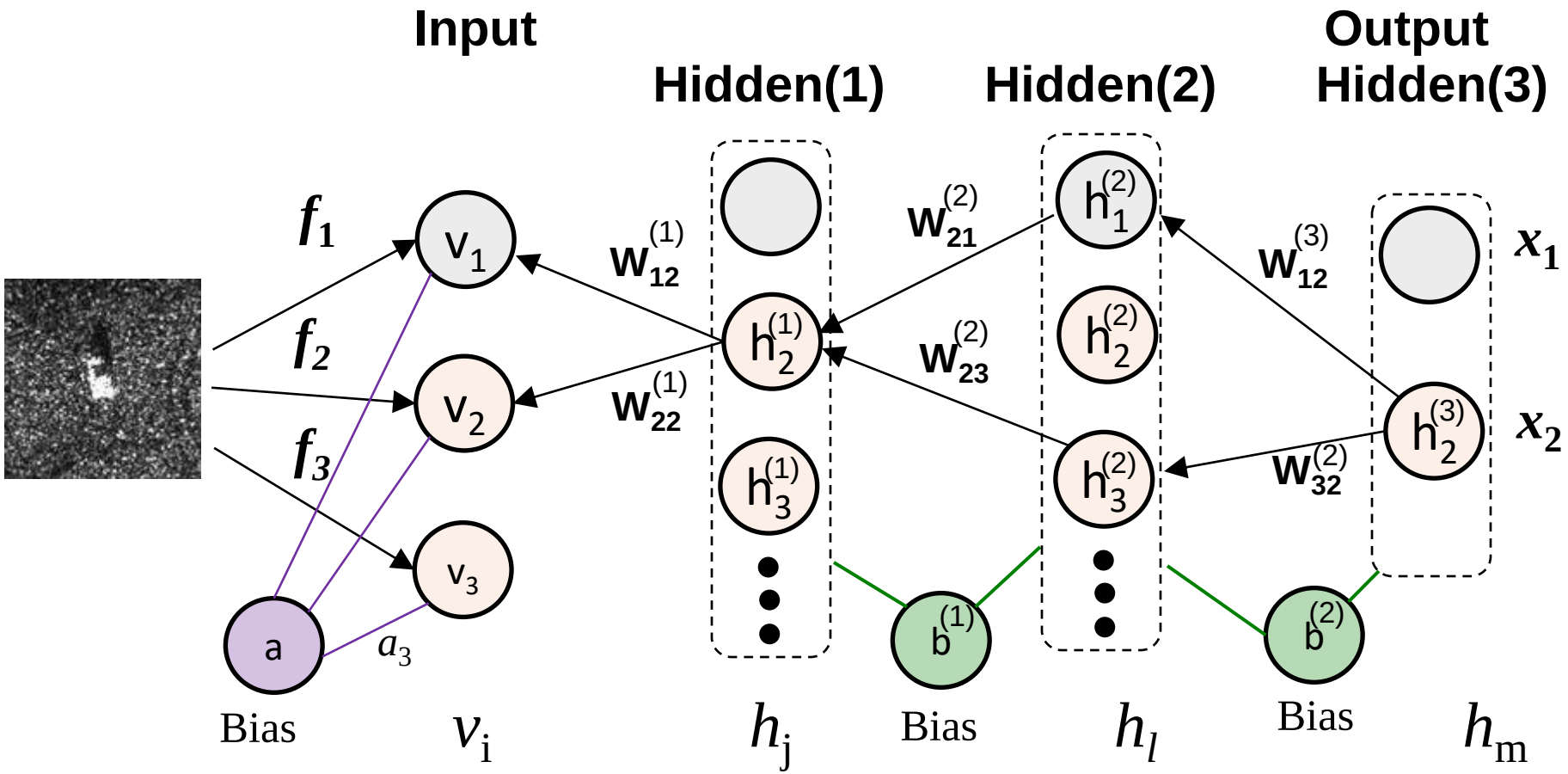
The difference in the global energy results from a single unit *i* equaling 0 (off) versus 1 (on)

$$\Delta E_i = \sum_{i<j} w_{ij} s_j + \sum_{i>j} w_{ji} s_j + \theta_i$$

- w_{ij} is the connection strength between unit *j* and unit *i*.
- s_i is the state, $s_i \in \{0,1\}$ of unit *i*, and
- θ_i is the bias of unit *i* in the global energy function, and $-\theta$ is the activation threshold for the unit.

Deep Boltzmann Machine

Figure 4.23



ADVANTAGE

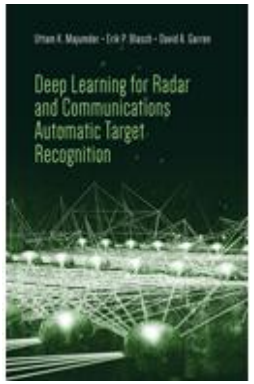
Learn complex relations from limited label data by doing processing in both directions

DISADVANTAGE

Slow as from the multiple MCMC steps

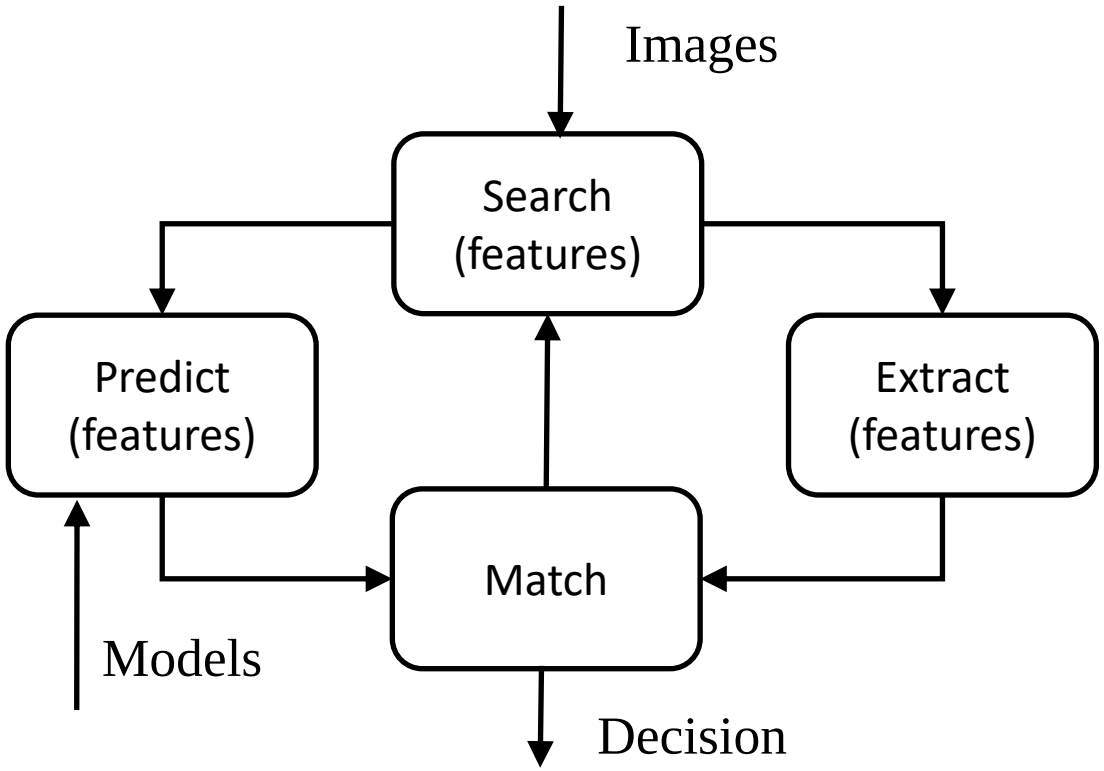
$$p(v) = \frac{1}{Z}\sum_{h} e^{\Sigma_{ij} W_{ij}^{(1)} v_i h_j^{(1)} + \Sigma_{jl} W_{jl}^{(2)} h_j^{(1)} h_l^{(2)} + \Sigma_{lm} W_{lm}^{(3)} h_l^{(2)} h_m^{(3)}}$$

Chapter 4 Outline

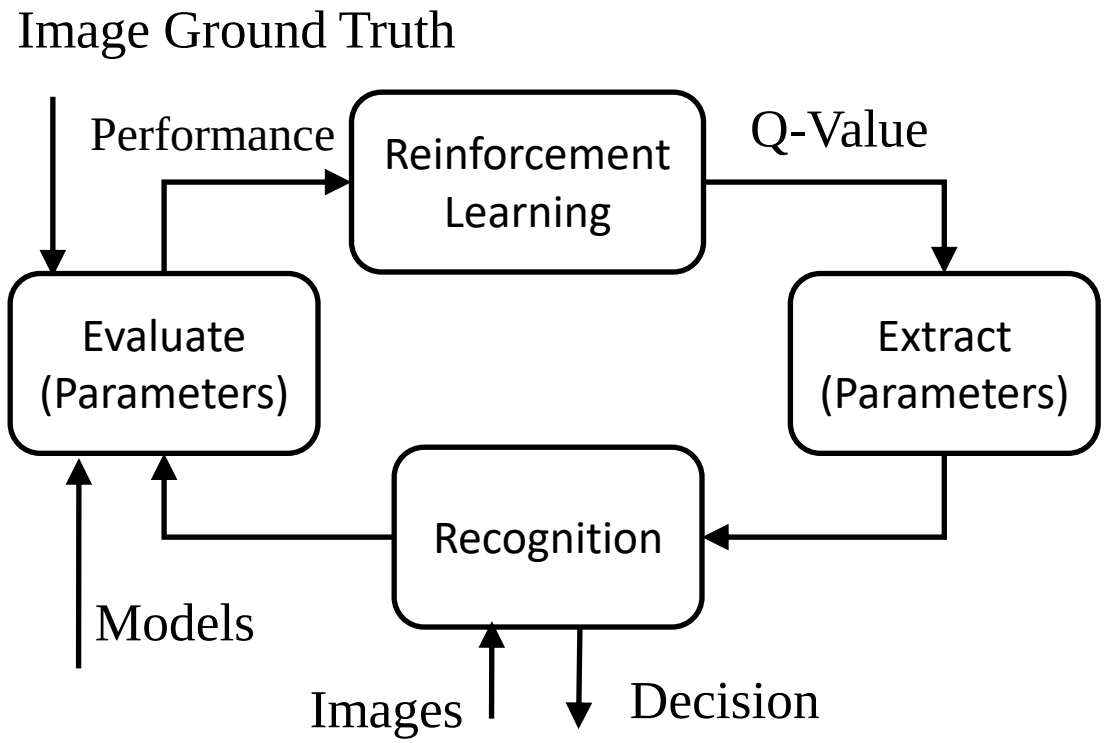


Learning Approaches

Figure 4.24



Model-based Learning



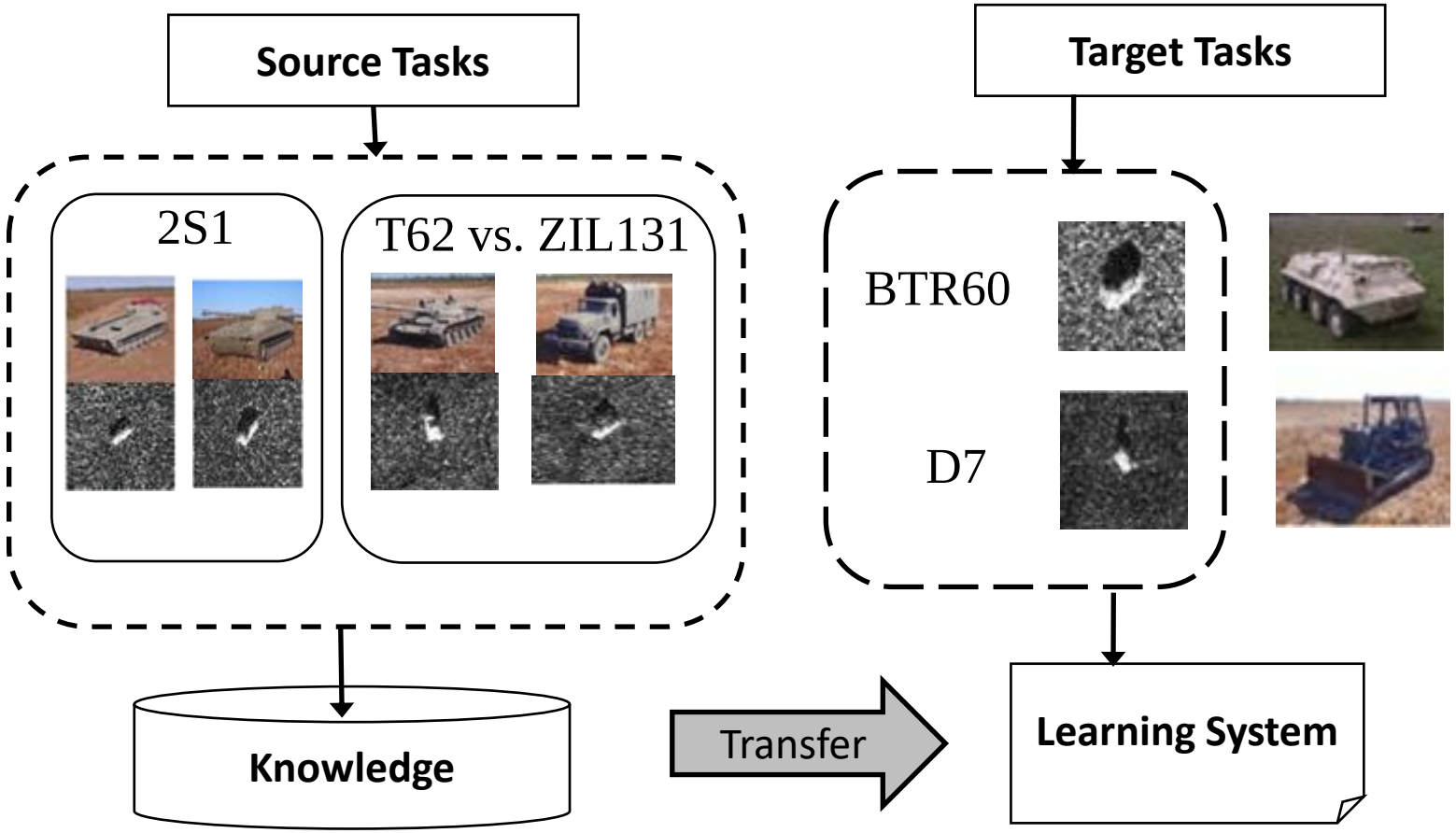
Reinforcement Learning

RL: Value function $V_\pi(s)$ describes the *expected return* starting with state $s_0 = s$, and **successively follows policy π.**

$$V_\pi(s) = E[R] = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,|s_0 = s\right]$$

Transfer Learning

Figure 4.25

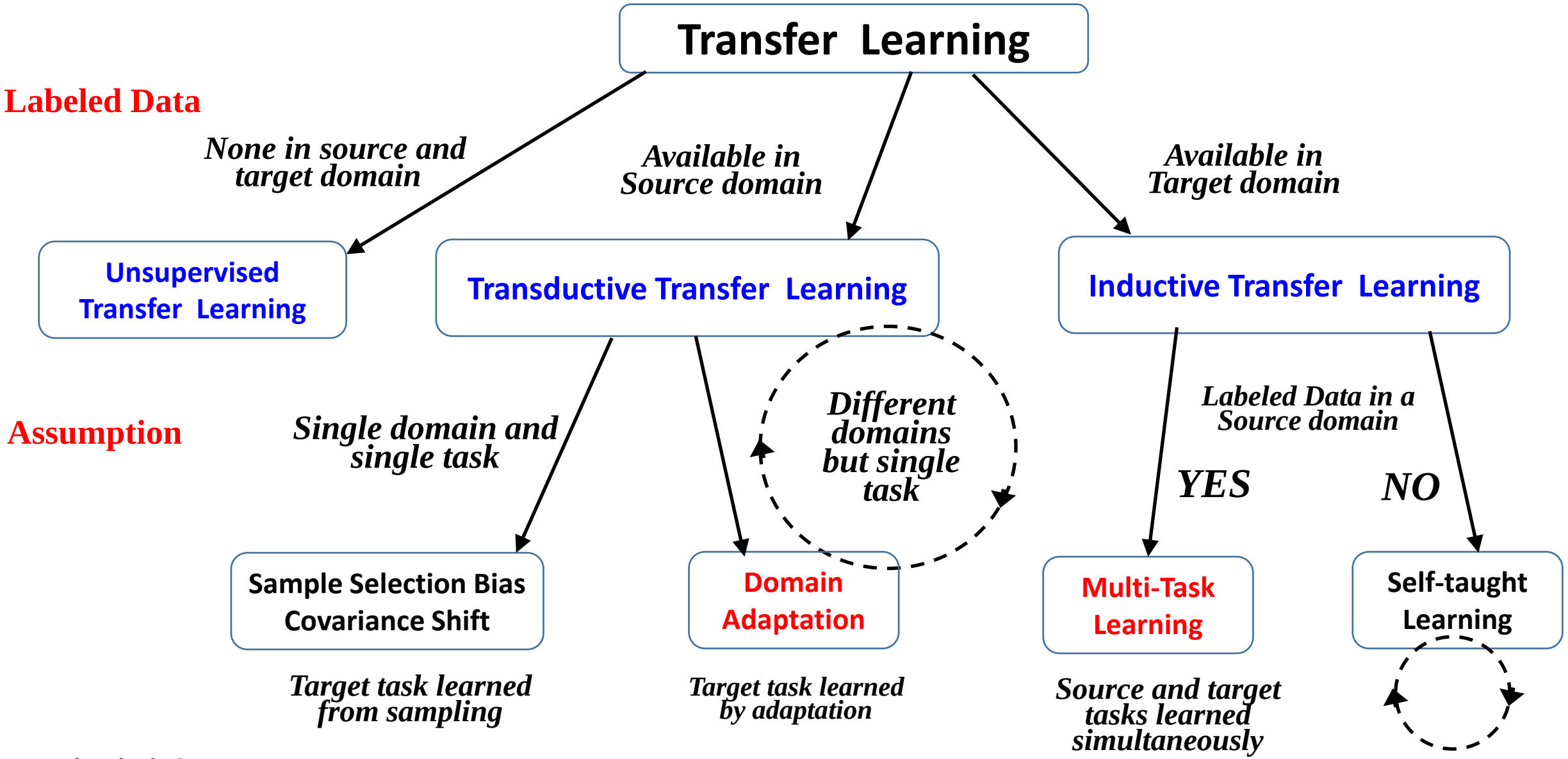


Transfer learning: Active Learning (Human to Machine)

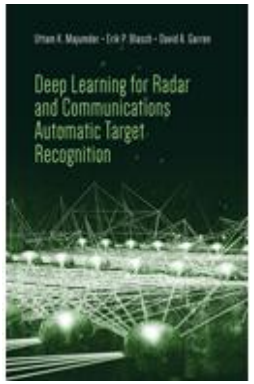
Domain adaptation (Machine to Machine)

Transfer Learning

Figure 4.26

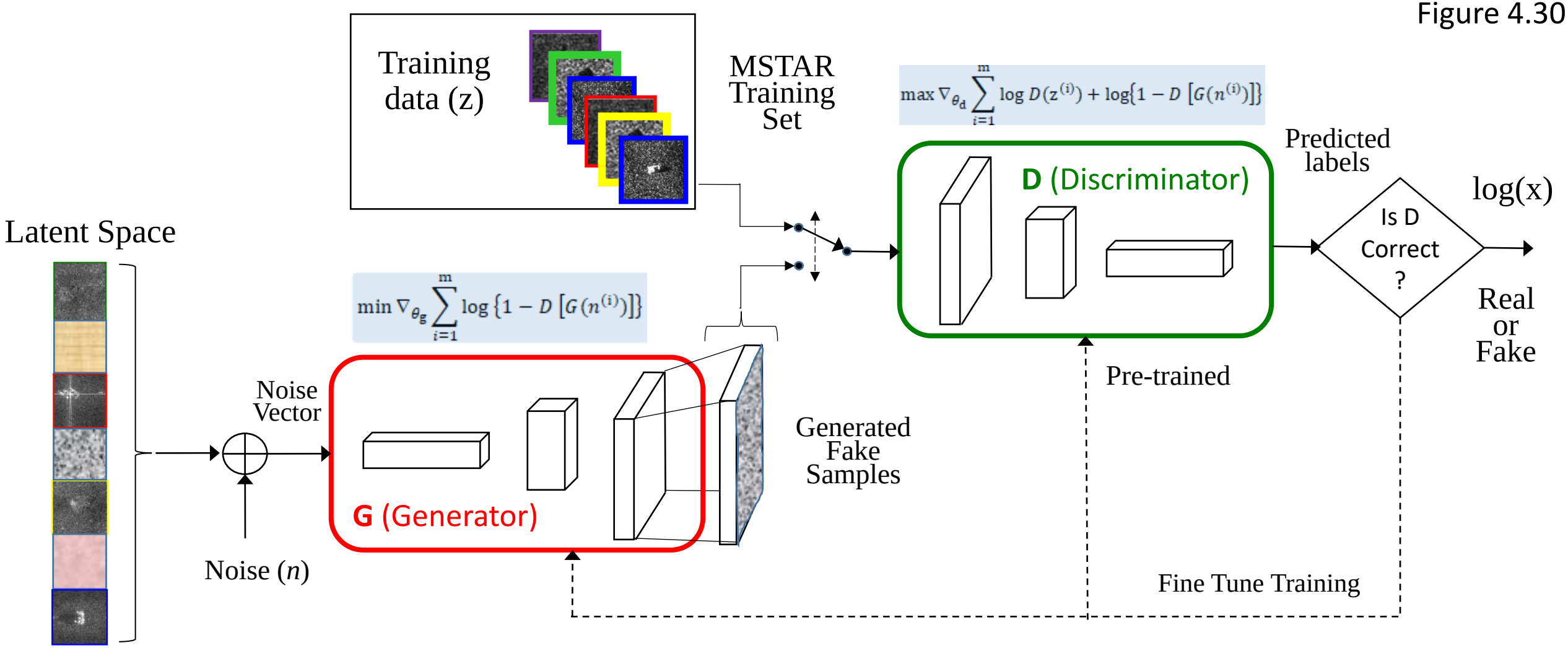


Chapter 4 Outline



Generative Adversarial Network

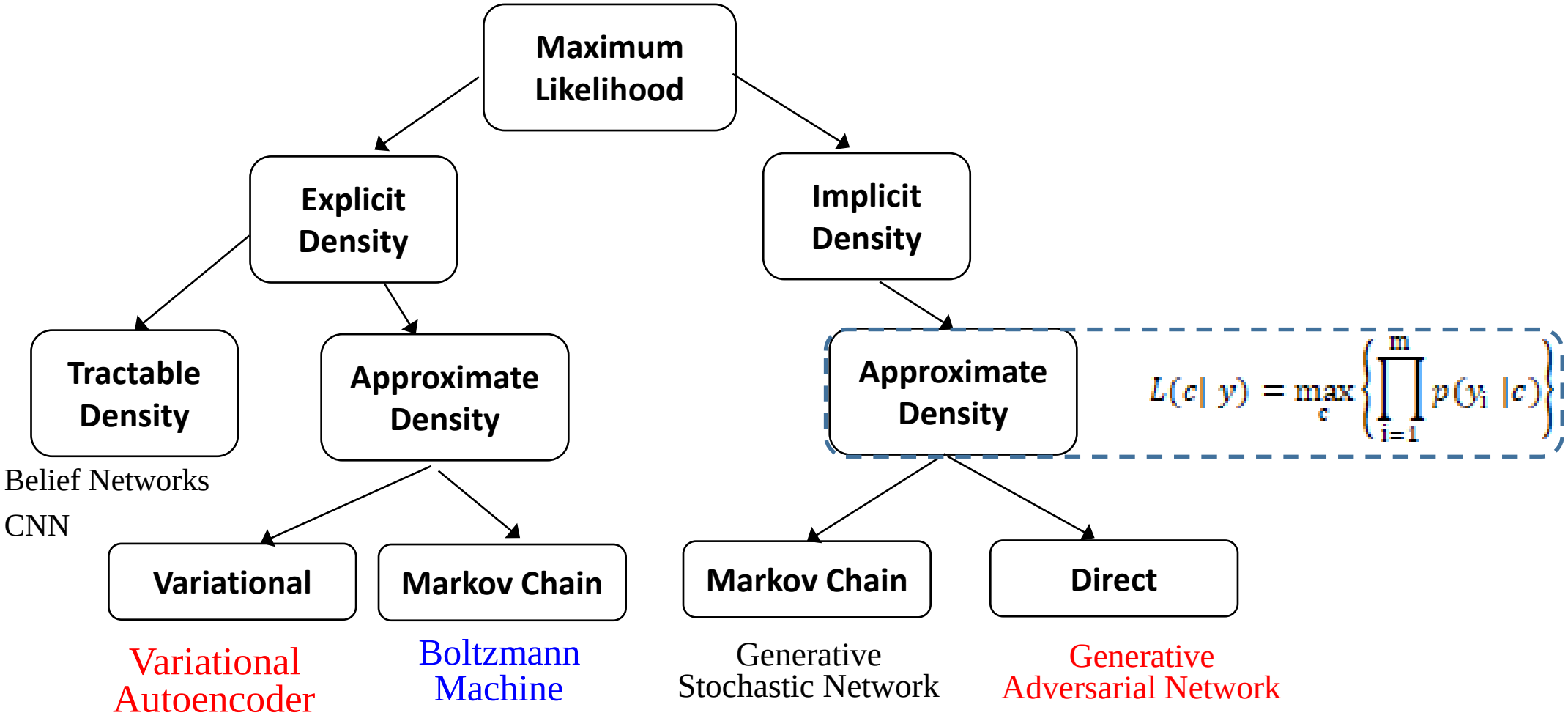
Figure 4.30



- In GANs, (uses an implicit transition operator to determine the decision density)
  - MLP ***generator*** **creates** new data instances,
  - MLP ***discriminator*** **evaluates** them for authenticity. That is. the discriminator decides if each instance of data under review belongs to the actual training dataset:

Generative Densities

Figure 4.29



Generative Adversarial Network (GAN)

Figure 4.27

- In GANs, (uses an implicit transition operator to determine the decision density)
  - MLP ***generator*** **creates** new data instances,
  - MLP ***discriminator*** **evaluates** them for authenticity. That is. the discriminator decides if each instance of data under review belongs to the actual training dataset:
- *Discriminative models* learn the boundary between classes, and
- *Generative models* model the distribution of individual classes

---

Deep generative models, such as graphical models,

- **Deep Belief Networks**: utilize **Bayesian reasoning** in a Markov Chain to determine $p(c|y) = [p(y \mid c)\, p(c)] / p(y)$; where $p(y \mid c)$ is the likelihood and $p(c)$ is the prior instances of a class.
- **Deep Belief Network (DBN)** : As Deep **Boltzmann machine** (with stochastic variables of the uncertainty and bias on c and y) is impossible to compute, so a **DBN is used for each layer**

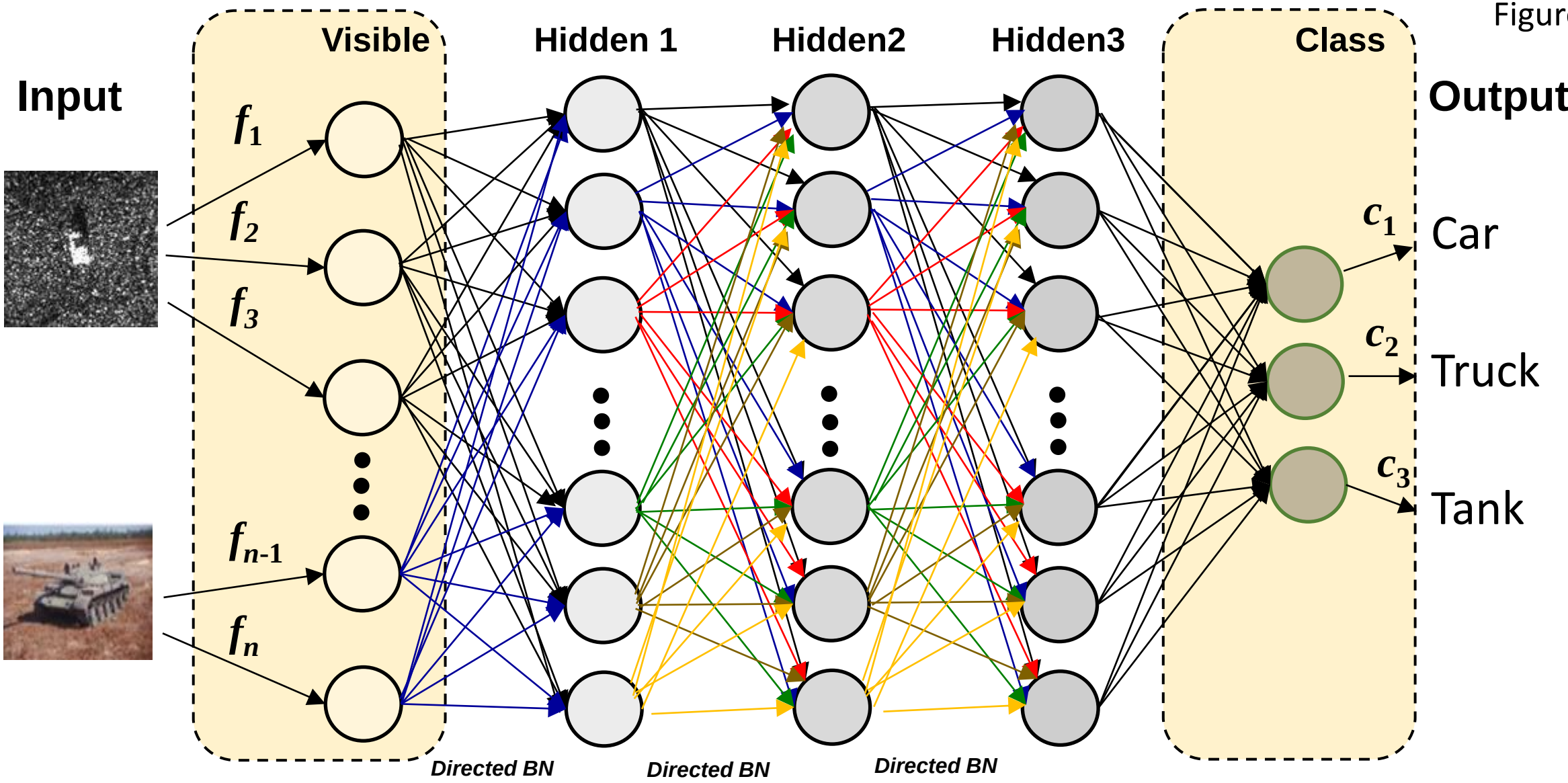
$$p(y, h^1, h^2, \ldots, h^n, c) = p(y|h^1)p(h^1|h^2) \ldots p(h^{n-2}|h^{n-1})p(h^{n-1}|h^n)\, p(h^n|c)\, p(c)$$

- The DBN gives tractable results; however, other methods discussed provide **approximate densities** $p(y \mid c)$ using the maximum likelihood.

$$L(c \mid y) = \max_c \left\{ \prod_{i=1}^{m} p(y_i \mid c) \right\}$$

Deep Belief Network

Figure 4.28



DBN is used for each layer: $p(y, h^1, h^2, \ldots, h^n, c) = p(y|h^1)p(h^1|h^2) \ldots p(h^{n-2}|h^{n-1})p(h^{n-1}|h^n)\, p(h^n|c)\, p(c)$

Generative Adversarial Network (GAN)

GAN Types:

- Given some of the **features**, they predict the associated features (GMM, Latent Dirichlet Allocation)
- Given a **label**, they predict the associated features (Naive Bayes, autoregressive), or
- Given a **hidden representation**, they predict the associated features (VAE, GAN)

Generator

$$\min \nabla_{\theta_g} \sum_{i=1}^{m} \log\{1 - D[G(n^{(i)})]\}$$

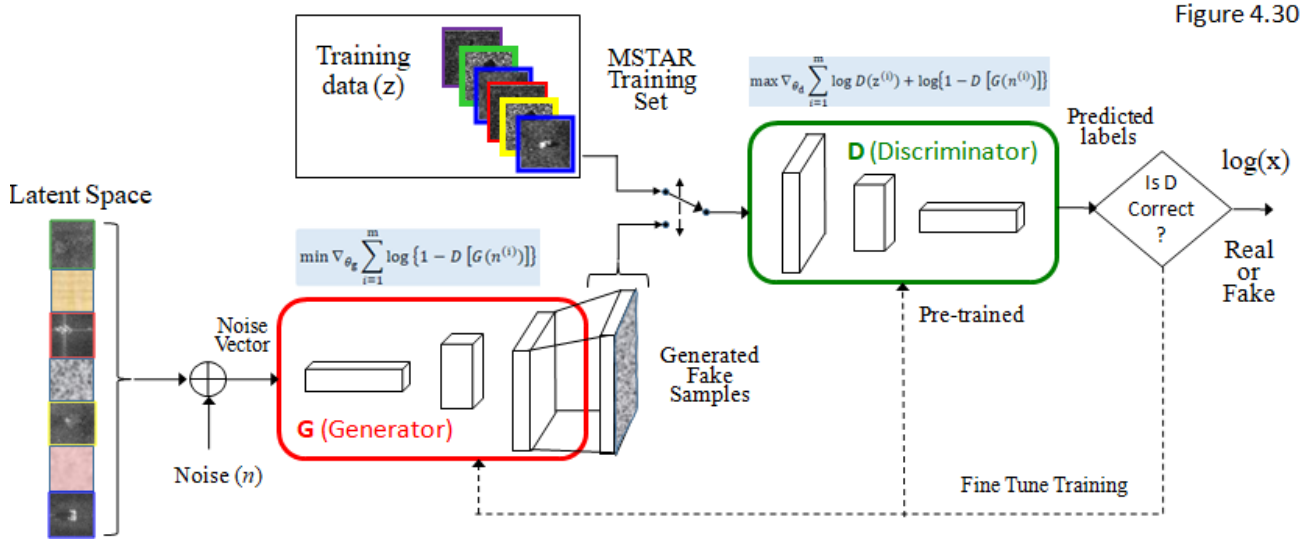
Discriminator

$$\max \nabla_{\theta_d} \sum_{i=1}^{m} \log D(z^{(i)}) + \log\{1 - D[G(n^{(i)})]\}$$

Min-Max Game

$$\min_G \min_D V(D,G) = \sum_{i=1}^{m} \log D(z^{(i)}) + \log\{1 - D[G(n^{(i)})]\}$$

Figure 4.30



GAN zoo (https://github.com/hindupuravinash/the-gan-zoo) provides a listing of available GANs

Generative Adversarial Network (GAN)

Comparisons (adapted from Goodfellow, *et al*., [**46**])

	Deep directed graphical models	Deep undirected graphical models	Generative Autoencoders	Adversarial Models
Model Design	Most models difficult	Careful design to ensure properties	Any differential function is theoretically permitted	Any differential function is theoretically permitted
Training	**Inference needed** during training	**Inference needed**. MCMC approximates partition function gradient	Enforced **tradeoff** between mixing and power of reconstruction generations	**Synchronizing** the discriminator with the generator
Inference	**Approximate** inference	**Variational** inference	**MCMC-based** inference	**Approximate** inference
Sampling	Available	Requires Markov Chain	Requires Markov Chain	Available
Evaluation p(z)	Intractable, but can be approximated	Intractable, but can be approximated	Not explicitly represented, may be approximated with Parzen density estimation	Not explicitly represented, may be approximated with Parzen density estimation

GAN zoo (https://github.com/hindupuravinash/the-gan-zoo) provides a listing of available GANs

Generative Adversarial Network (GAN)

Steps:

Step 1: The generator takes in **random numbers** and returns an image.

Step 2: The **generated image (data)** is fed into the discriminator alongside a stream of images taken from the actual, ground-truth dataset.

Step 3: The **discriminator injects both real and fake images and returns probabilities**, each being a number between 0 and 1, with

1 representing a prediction of **authenticity**,
0 representing the **imposter**.

ALGORITHM: GAN

INPUT: number of iterations k, MLP with parameters θ_g

INITIALIZE:

- Noise samples m: $\mathbf{n} = \{n^{(1)}, \ldots, n^{(m)}\}$ from prior $p_g(n)$
- Data samples m : $\mathbf{d} = \{z^{(1)}, \ldots, z^{(m)}\}$ from generating distribution $G(n; \theta_g)$

For $i = 1, 2, \ldots, k$

- receive instance: $\mathbf{d}_t \in \mathbb{R}^n$, $\mathbf{n}_t \in \mathbb{R}^n$
- update the discriminator by ascending its stochastic gradient

$$\max \nabla_{\theta_d} \sum_{i=1}^{m} \log D(z^{(i)}) + \log\{1 - D[G(n^{(i)})]\}$$

- receive instance: $\mathbf{n}_t \in \mathbb{R}^n$
- update the generator by descending its stochastic gradient

$$\min \nabla_{\theta_g} \sum_{i=1}^{m} \log\{1 - D[G(n^{(i)})]\}$$

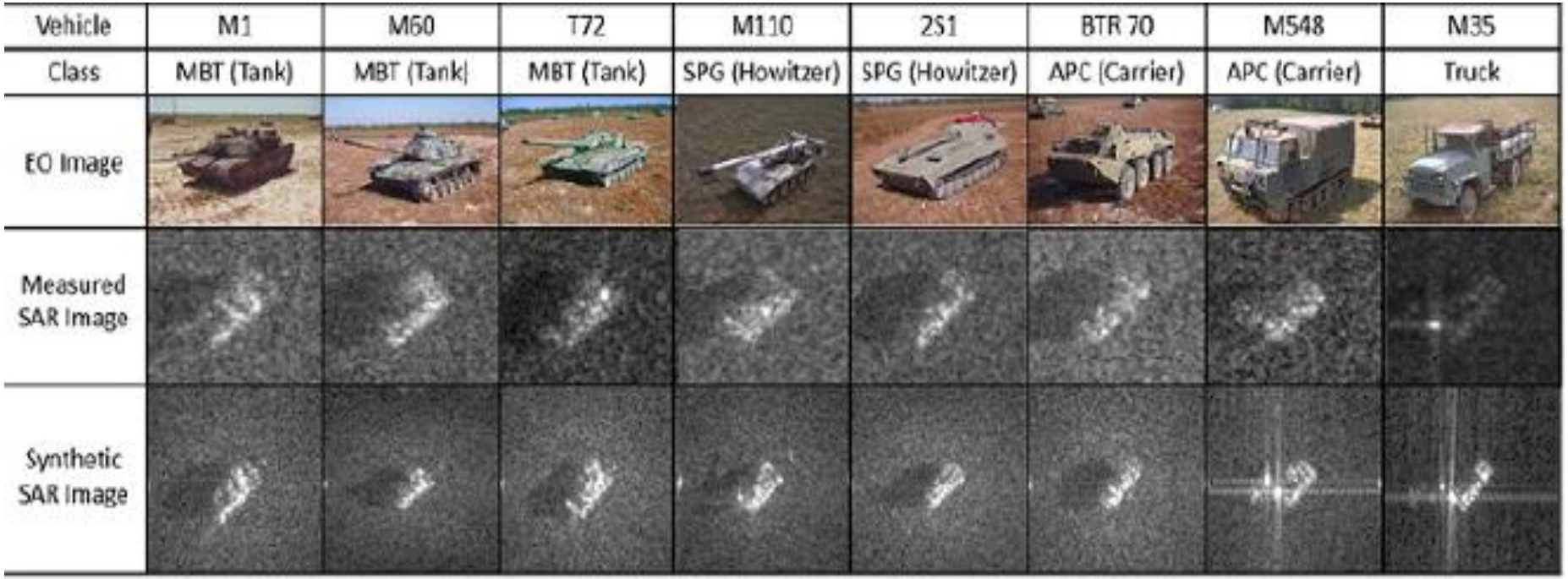
- output value function and data z

GAN zoo (https://github.com/hindupuravinash/the-gan-zoo) provides a listing of available GANs

EO with SAR Image

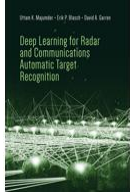
Figure 4.27

- GAN Example

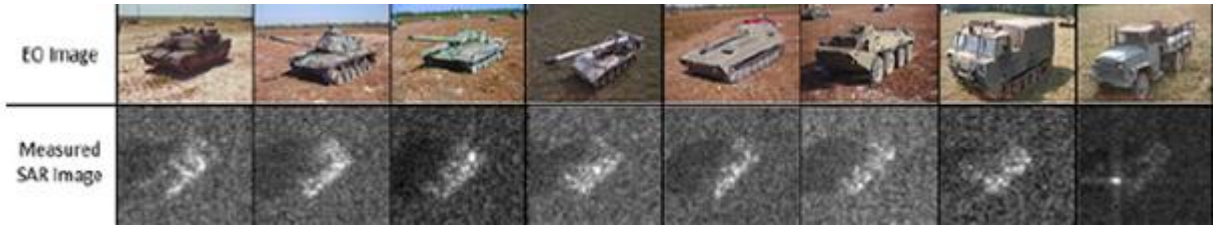


- From Arnold, J., L. Moore, and E. Zelnio. "Blending **Synthetic** and **Measured** Data Using Transfer Learning for Synthetic Aperture Radar (SAR) Target Classification," *Proc. SPIE, Vol.*10647, 2018.

Takeaway Message



Method	Performance (%)			Reference
	SOC	EOC-1	EOC-2	
Pulse Couple Neural Network (PCNN) [35 tgts]	87.0	75	67	Blasch, 1998
A-ConvNets	99.1	96.1	98.9	Chen, *et al*., 2016
AFRLeNet	**99.4**			**Profeta *et al*., 2016**
CNN-SVM	99.5	95.75		Amrani, *et al*., 2017
VGG-S1	98.8	94.15		Amrani, *et al*., 2017
VGG-S, Feature Fusion, KNN	99.82	96.16		Amrani, *et al*., 2017
Multi-aspect aware bidirectional LSTM RNN	99.90	-	99.59	Zhang, *et al*. 2017
Joint Supervised Dictionary	97.65	98.39		
GAN-CNN	97.53			Zheng, C et al, 2019
MGAN-CNN	97.81			Zheng, C et al, 2019
Triple GAN and Integrated GAN	99.9			Gao, F. Et al 2019



Chapter 4 Outline