



$\mathcal{X}$-metric: An N-Dimensional Information-Theoretic Framework for Groupwise Registration and Deep Combined Computing

Xin Gao

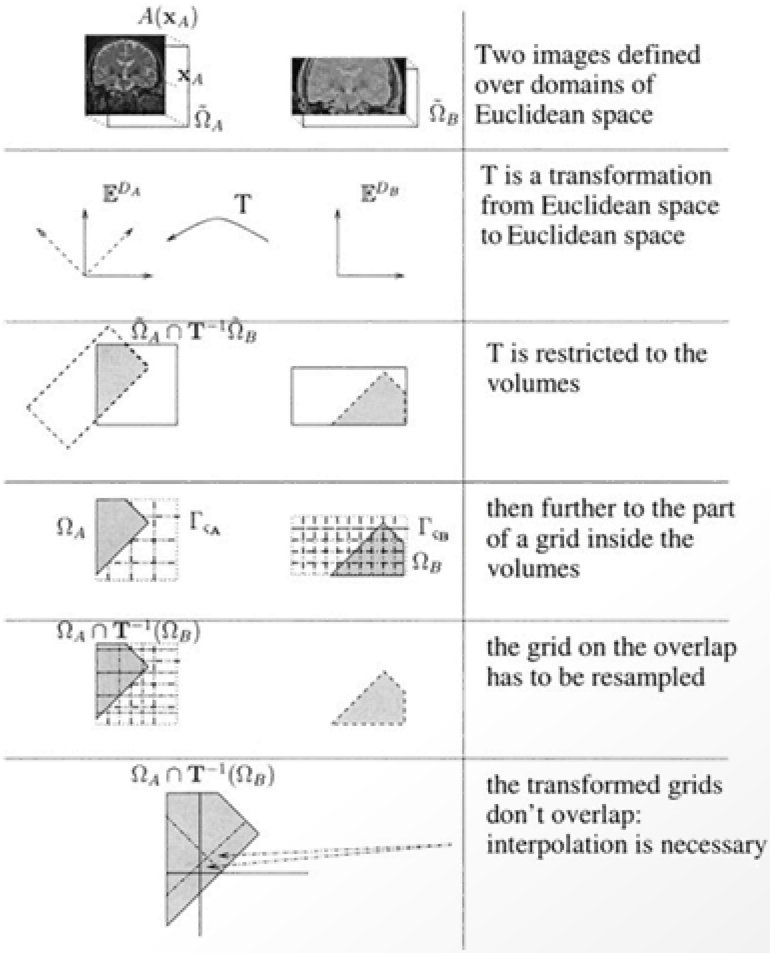
2023.12.21

X. Luo and X. Zhuang, "$\mathcal {X}$-Metric: An N-Dimensional Information-Theoretic Framework for Groupwise Registration and Deep Combined Computing," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 45, no. 7, pp. 9206-9224, 1 July 2023, doi: 10.1109/TPAMI.2022.3225418.

Content

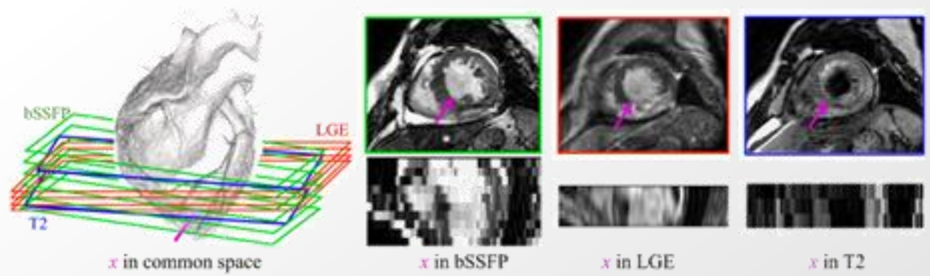
1.1 Group Registration

Objective: Recover the spatial correspondences of two or multiple images by maximizing a given similarity metric.



Pairwise

Groupwise



The common space ?

- Unbiased groupwise Registration
- Group-to-reference Registration

Hill, Derek LG, et al. "Medical image registration." *Physics in medicine & biology* 46.3 (2001): R1.

1.1 Group Registration

大数据学院 School of Data Science

Main concepts and notations:

- Image space Ω_j , Spatial samples $\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{\xi} \in \Omega_j$
- The observed image group $\boldsymbol{U} = \{U_j\}_{j=1}^{N}$, $U_j\colon \Omega_j \to \mathbb{R}$
- The j-th image $U_j = (u_{j\omega})_{\omega\in\Omega_j}$, $u_{j\omega}$ abbreviation for $U_j(\boldsymbol{\omega})$, where $\boldsymbol{\omega} \in \boldsymbol{\Omega_j}$
- Common space / Common coordinate system Ω
- Spatial transformation $\boldsymbol{\phi} = \{\phi_j\}_{j=1}^{N}$, $\phi_j : \Omega \to \Omega_j$
- The resampled intensity vector $\boldsymbol{u}_{\boldsymbol{x}}^{\boldsymbol{\phi}} = [u_{\boldsymbol{x},1}^{\phi_1}, \ldots, u_{\boldsymbol{x},N}^{\phi_N}]^T$, where $u_{x,j}^{\phi_j} \triangleq U_j \circ \phi_j(\boldsymbol{x})$, $\boldsymbol{x} \in \boldsymbol{\Omega}$

The purpose of co-registration / group registration:

Given N observed image group $\boldsymbol{U} = \{U_j\}_{j=1}^{N}$

Find the spatial transformation $\boldsymbol{\phi} = \{\phi_j\}_{j=1}^{N}$ that aligns them into a common coordinate system Ω.

1.1 Group Registration

Joint Intensity Distribution

$$P_\phi(\boldsymbol{U}) = \prod_{\boldsymbol{x}\in\Omega} P\left(\boldsymbol{u}_{\boldsymbol{x}}^{\phi}; \alpha(\boldsymbol{x})\right)$$

which is factorized over $i.i.d$ spatial samples $\boldsymbol{x} \in \boldsymbol{\Omega}$, $\alpha(\boldsymbol{x})$ is the parameter of the distribution for every intensity vector, which can be spatially variant.

Maximum likelihood approach

Find the optimal spatial correspondences through the MLE of a multivariate JID indexed by the spatial transformation $\boldsymbol{\phi}$

$$\mathcal{L}\left(\theta \mid \boldsymbol{u}_{\boldsymbol{x}}^{\phi}\right) = \sum_{\boldsymbol{x}\in\Omega} \log P\left(\boldsymbol{u}_{\boldsymbol{x}}^{\phi}; \alpha(\boldsymbol{x})\right)$$

Parameters concerned $\boldsymbol{\theta}$:

- ✓ spatial transformation: $\boldsymbol{\phi}$
- ✓ distribution parameter: $\alpha(\boldsymbol{x})$

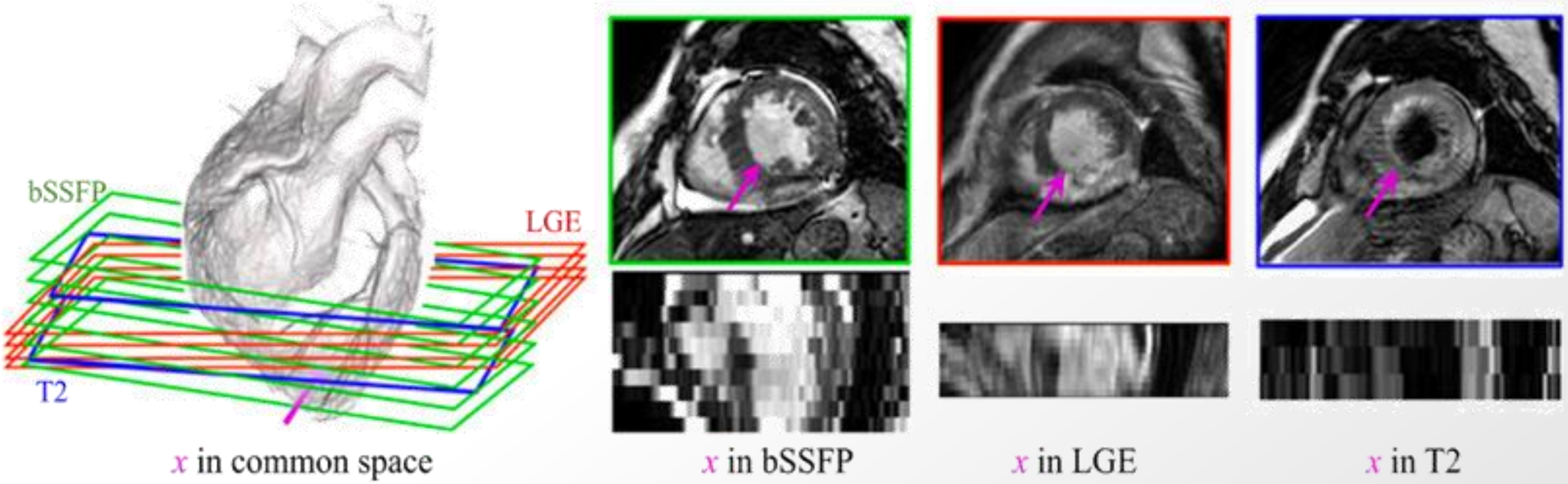
1.1 Combined computing

大数据学院
School of Data Science

Combining registration with segmentation in a unified framework

Medical images are usually complementary yet inherently correlated through their underlying **common anatomy**.

Common space / Common coordinate system / Common anatomy



Categorical latent variables

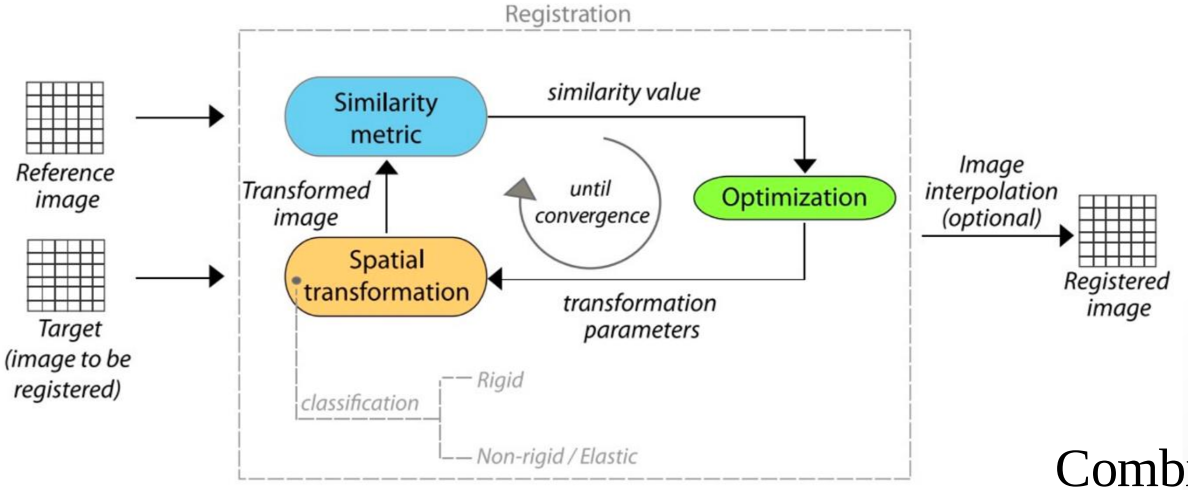
Generative model - GMM

$$P(\boldsymbol{U} \mid \boldsymbol{Z}) = \prod_{j=1}^{N} P(U_j \mid \boldsymbol{Z})$$

1.1 Combined computing

大数据学院
School of Data Science

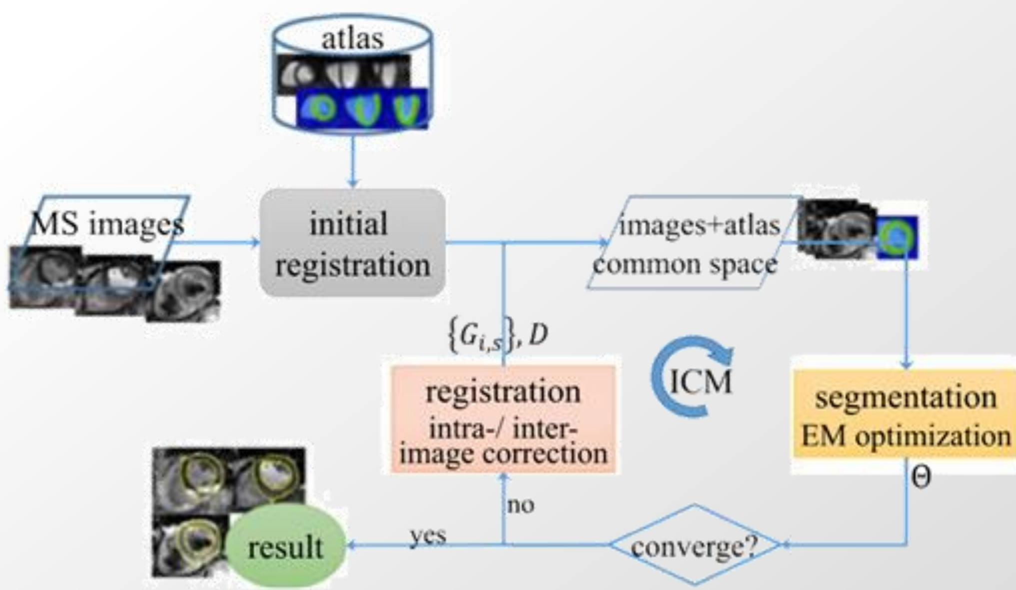
Registration



Registration

- **Similarity metric**
- **Spatial transformation**

Combined computing in MvMM



Segmentation

- **Common anatomy**
- **Spatial transformation**

Note: Assume that the anatomical structures in the image can be entirely distinguished based on the pixel values.

1.2 $\mathcal{X}$-metric

- **What?**

A groupwise similarity metric

$$\mathcal{X}(\boldsymbol{U},\boldsymbol{Z}) = \sum_{j=1}^{N} I(U_j,\boldsymbol{Z}) = \sum_{j=1}^{N}[H(U_j) + H(\boldsymbol{Z}) - H(U_j,\boldsymbol{Z})]$$

- **How?**

The statistical dependency of a set of random variables

$$C(\boldsymbol{U}) \triangleq D_{\mathrm{KL}}\left[P(\boldsymbol{U})\| \prod_{j=1}^{N} P(U_j)\right] = \left[\sum_{j=1}^{N} H(U_j)\right] - \boxed{H(\boldsymbol{U})}$$

+

The intensity-class mutual information

$$I(\boldsymbol{U},\boldsymbol{Z}) = H(\boldsymbol{U}) - H(\boldsymbol{U}\mid\boldsymbol{Z}) = \boxed{H(\boldsymbol{U})} - \sum_{j=1}^{N} H(U_j\mid\boldsymbol{Z})$$

- **Why?**

✓ It can measure the statistical dependency among an arbitrary number of images.

✓ The computation of the joint entropy $H(U)$ is computationally prohibitive in general for $N \gg 2$.

大数据学院 School of Data Science

- **What?** A generic co-registration algorithm

- **How?**

$$\hat{\boldsymbol{\phi}} = \arg\max_{\boldsymbol{\phi}} \max_{\boldsymbol{\alpha}} \mathcal{X}(\boldsymbol{U}[\boldsymbol{\phi}], \boldsymbol{Z})$$

Common space parameters: $\alpha = \{\pi, \ \Gamma\}$

$$\boldsymbol{\alpha}^{[t+1]} = \arg\max_{\boldsymbol{\alpha}} \mathcal{X}\left(U\left[\boldsymbol{\phi}^{[t]}\right], Z\right)$$

Transformation parameters: $\phi = \{\phi_j\}_{j=1}^{N}$

$$\boldsymbol{\phi}^{[t+1]} = \arg\max_{\boldsymbol{\phi}} \mathcal{X}\left(U[\boldsymbol{\phi}], \boldsymbol{Z}^{[t+1]}\right)$$

- **Why?**

✓ No closed-form solution of the inner optimization, coordinate ascent

✓ **Maximum log-likelihood and EM insights**

1.2 The proposed framework

大数据学院
School of Data Science

- A generic probabilistic framework for **estimating the statistical dependency** and **finding the anatomical correspondences** among an **arbitrary number** of medical images.
- **$\mathcal{X}$-metric**: Information-theoretic metric
- **$\mathcal{X}$-CoReg**: Co-registration algorithm
- ***N***: Groupwise registration of the *N* observed images
- Extended to Deep Combined Computing

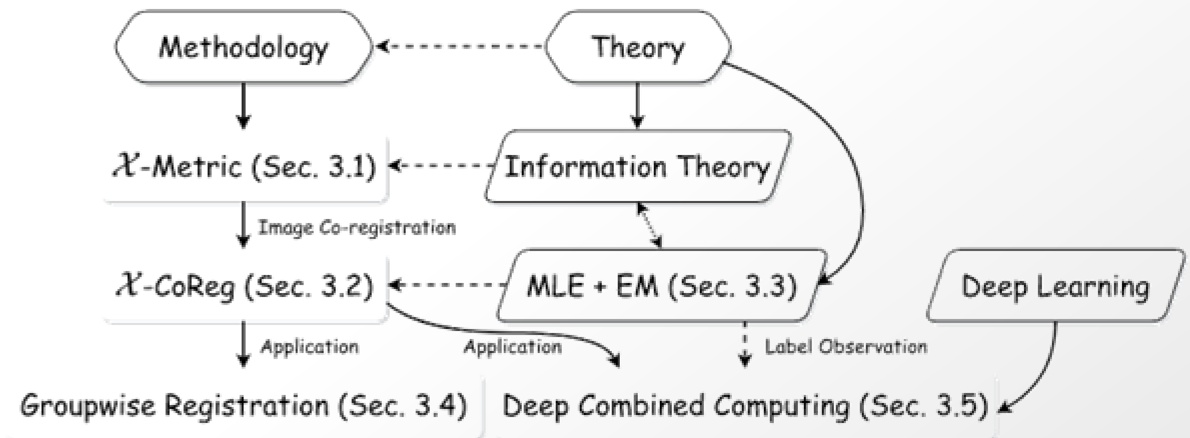


Fig. 1. Roadmap of the proposed framework.

It can be interpreted from both the information-theoretic and the MLE perspective

Content

大数据学院
School of Data Science

- 1 Introduction to the framework (15mins)
  - 1.1 Group Registration & Deep combined computing (10mins)
  - 1.2 $\mathcal{X}$-metric and $\mathcal{X}$-CoReg (5mins)
- 2 Preliminaries (30mins)
  - 2.1 Entropy and Mutual information (MI) (10mins)
  - 2.2 EM algorithm and combined computing (20mins)
- 3 Generic Framework for Registration (35mins)
  - 3.1 Notation and Graphic representation (10mins)

Break (5 mins)

  - 3.2 MLE insights and EM (20mins)
  - 3.3 $\mathcal{X}$-metric and $\mathcal{X}$-CoReg (5mins)
- 4 Extended Framework for DeepCC (35mins)
  - 4.1 Graphic representation and Framework Modification (5mins)
  - 4.2 MLE => Loss function (15mins)
  - 4.3 Network Architecture and training Pipeline (15mins)
- 5 Experiment (15 mins)

2.1 Entropy

大数据学院
School of Data Science

Information

- Given a discrete random variable X with probability distribution p(x), its information is defined as

$$h(x) = -\log p(x)$$

Shannon's entropy

- Definition: Given events $e_1, \ldots, e_m$ occurring with probabilities $p_1, \ldots, p_m$, the Shannon's entropy is defined as

$$H(X) = \sum_i p_i \log \frac{1}{p_i} = -\sum_i p_i \log p_i$$

- Interpretations:
  - The amount of average information
  - The uncertainty of the random variable
  - The dispersion of the probability distribution

2.1 Entropy

Joint Entropy

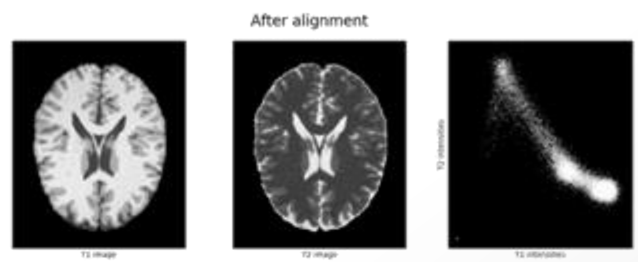
- **Definition:** Given random variables $X_1, \dots, X_n$ and their joint distribution $p(x_1, \dots, x_n)$, the joint entropy of $X_1, \dots, X_n$ is defined as

$$H(X_1, \dots, X_n) = -\sum_{x_1} \dots \sum_{x_n} p(x_1, \dots, x_n) \log p(x_1, \dots, x_n)$$

- **Interpretations:**

Joint histogram

When the images are correctly registered, the joint histogram shows certain clusters for the gray values of anatomical structures.



Joint histogram

As the images become misaligned, the joint intensity histogram displays a dispersion of the clustering.

  - The dispersion of the clustering.
  - A groupwise similarity metric. By finding the transformation that minimizes the joint entropy, images should be registered.
- **Drawbacks:** when $n \gg 2$, it can be computationally prohibitive.

2.1 Mutual information (MI)

大数据学院 School of Data Science

Definition:

- For two random variables U and Z, the mutual information can be defined as:

$$I(U, Z) = H(U) - H(U|Z) = H(Z) - H(Z|U)$$

- MI can be related to the joint entropy in the sense:

$$I(U, Z) = H(U) + H(Z) - H(U, Z)$$

Maximum & minimum:

- The maximum attains when U is totally dependent on Z.
- The minimum attains when U and Z are independent.

Multivariate random variable:

- If $\boldsymbol{U} = (U_j)$, U_j are assumed conditionally independent given Z, $j = 1, \ldots, N$, i.e., $P(\boldsymbol{U}|Z) = \prod_{j=1}^{N} P(U_j|Z)$,
- then MI becomes $I(\boldsymbol{U}, Z) = H(\boldsymbol{U}) - \sum_{j=1}^{N} H(U_j|Z)$.

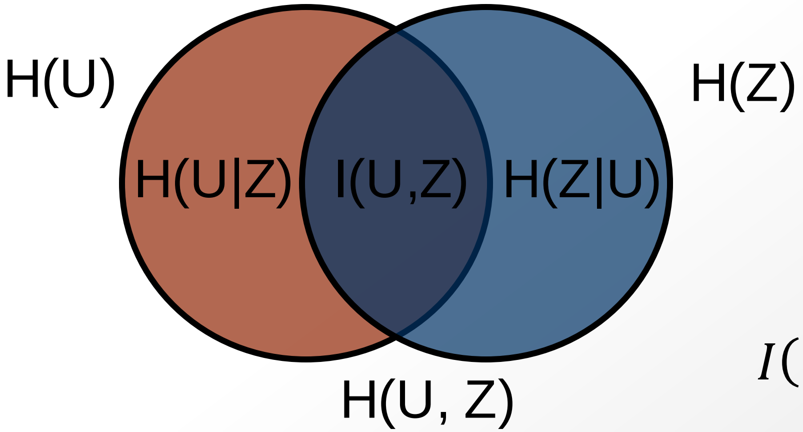
2.1 Mutual information (MI)

Interpretation

- A metric which measures the similarity/dependency between U and Z.
- The reduction of the amount of uncertainty about one random variable when the other one is known.

The advantage of MI over JE in registration:

- The marginal entropies will have low values when the overlapping part of the images contains only background and high values when it contains anatomical structures.



$$I(U,Z) = H(U) - H(U|Z)$$

$$I(U,Z) = H(Z) - H(Z|U)$$

$$I(U,Z) = H(U) + H(Z) - H(U,Z)$$

$$I(U,Z) = H(U,Z) - H(U|Z) - H(Z|U)$$

Pluim, Josien PW, JB Antoine Maintz, and Max A. Viergever. "Mutual-information-based registration of medical images: a survey." *IEEE transactions on medical imaging* 22.8 (2003): 986-1004.

2.2 EM algorithm

大数据学院 School of Data Science

- Complete data: U, Z
  - Observed data, U: intensity in the image, $U = \{u_x\}_{x\in\Omega}$
  - Latent variable, Z: common anatomy, the label in the common space, $P(z_x = k) = \pi_k$, $\sum_{k=1}^{K} \pi_k = 1$
  - Assume: $U_x | Z_x = k \sim f_k(u_x; \theta_k), k = 1, 2, \dots, K$
- Parameters
  - Categorical prior, π
  - Appearance model's parameter, θ
- Joint distribution and marginal:

$$P(u_x, Z_x \mid \theta) = \prod_{k=1}^{K} \left(P(u_x, Z_x = k \mid \theta)\right)^{1(Z_x=k)} = \prod_{k=1}^{K} \left(\pi_k f_k(u_x; \theta_k)\right)^{1(Z_x=k)}$$

$$P(u_x \mid \theta) = \sum_k P(u_x, Z_x = k \mid \theta) = \sum_k P(Z_x = k \mid \theta) P(u_x \mid Z_x = k, \theta) = \sum_k \pi_k f_k(u_x; \theta_k)$$

Maximize the likelihood of observed data, i.e., $\log P(U|\theta)$

$$\log P(u_x \mid \theta) = \log P(u_x, Z_x \mid \theta) - \log P(Z_x \mid u_x, \theta)$$

$$\Downarrow$$

Compute and maximize Q function

2.2 EM algorithm

- Q function: $$Q\left(\theta \mid \theta^{[t]}\right) = E_{Z|U,\theta^{[t]}} \log p(U, Z; \theta)$$
$$= \mathbb{E}\left\{\log P(u, Z \mid \theta) \mid u, \theta^{[t]}\right\}$$
$$= \mathbb{E}\left\{\sum_{x\in\Omega}\sum_{k=1}^{K} 1(Z_x = k)(\log \pi_k + \log f_k(u_x; \theta_k)) \mid u, \theta^{[t]}\right\}$$
$$= \sum_{x\in\Omega}\sum_{k=1}^{K} \mathbb{E}\left\{1(Z_x = k) \mid u, \theta^{[t]}\right\}(\log \pi_k + \log f_k(u_x; \theta_k))$$
$$= \sum_{x\in\Omega}\sum_{k=1}^{K} P\left\{Z_x = k \mid u, \theta^{[t]}\right\}(\log \pi_k + \log f_k(u_x; \theta_k))$$
$$= \sum_{x\in\Omega}\sum_{k=1}^{K} q_{xk}^{[t]}(\log \pi_k + \log f_k(u_x; \theta_k))$$

where

$$q_{xk}^{[t]} = P\left(Z_x = k \mid u_x, \theta^{[t]}\right) = \frac{P\left(Z_x = k \mid \theta^{[t]}\right) P\left(u_x \mid z_x = k, \theta^{[t]}\right)}{\int_l P\left(Z_x = l \mid \theta^{[t]}\right) P\left(u_x \mid z_x = l, \theta^{[t]}\right)} = \frac{\pi_k^{[t]} f_k^{[t]}\left(u_x; \theta_k^{[t]}\right)}{\Sigma_{l=1}^{K} \pi_l^{[t]} f_l^{[t]}\left(u_x; \theta_l^{[t]}\right)}$$

⇨ **Estimates of the parameters are given to maximize the Q function, or to increase the value of the Q function**

2.2 EM algorithm

大数据学院 School of Data Science

$$Q\left(\theta \mid \theta^{[t]}\right)=\sum_{x \in \Omega} \sum_{k=1}^{K} \frac{\pi_{k}^{[t]} f_{k}^{[t]}\left(u_{x} ; \theta_{k}^{[t]}\right)}{\sum_{l=1}^{K} \pi_{l}^{[t]} f_{l}^{[t]}\left(u_{x} ; \theta_{l}^{[t]}\right)}(\underline{\log \pi_{k}}+\underline{\log f_{k}\left(u_{x} ; \theta_{k}\right)})$$

- **Objective:** maximize/ increase Q function
- **Estimate the Q function**
  - Estimate prior π
  - Estimate parameters in the appearance model
- **Methods**
  - If parameters can be solved **analytically**, just let the derivative equal to zero.
  - If parameters can not be solved analytically, **numerical methods** can be used to give estimations and increase the value of Q function

2.2 EM algorithm

Iterations given by EM algorithm

- $q_{xk}^{[t]} = p(Z_x = k|\, u_x, \theta^{[t]}) = \frac{\pi_k^{[t]} f_k^{[t]}(u_x; \theta_k^{[t]})}{\sum_{l=1}^{K} \pi_l^{[t]} f_l^{[t]}(u_x; \theta_l^{[t]})}$
- $\pi_k^{[t+1]} = \frac{1}{|\Omega|} \sum_x q_{xk}^{[t]}$

Summation of EM algorithm

- Initialize distribution parameters
- Repeat the following two steps until convergence:
  - E-step: Compute the posterior and Q function
  - M-step: Maximize the Q function (or increase the value of the Q function)

2.2 Combined computing

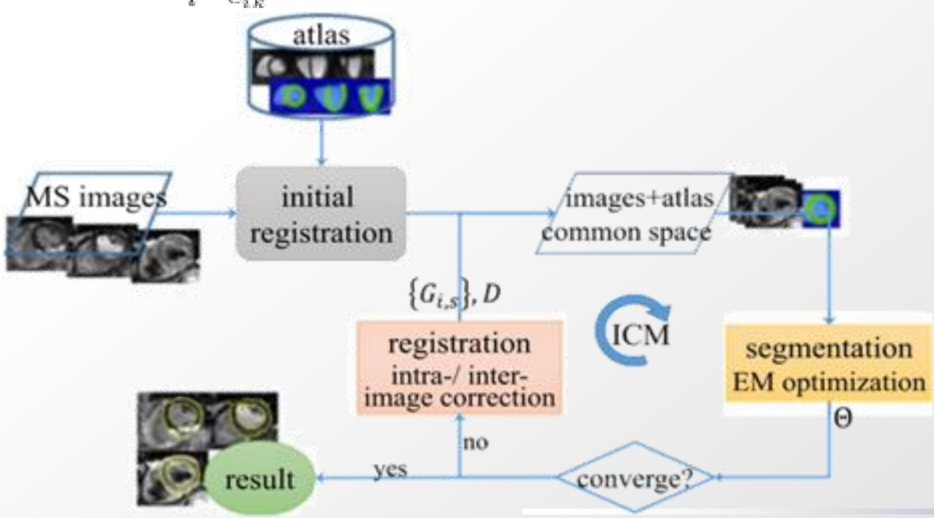
- How mixture model and EM algorithm can be used to complete segmentation and registration task

By update common space parameter π: gain anatomy structure in the common space ⟹ Achieve segmentation

Introduce parameters:
Affine transformations $\{G_{i,s}\}$, atlas deformation D

By update $\{G_{i,s}\}$ and D ⟹ Achieve transformation

$$
\begin{aligned}
& LL(\theta, D, \{G_{i,s}\}) \\
&= \sum_{x\in\Omega} \log\left\{\sum_k p(s(x)=k|D)\prod_i \sum_{c_{ik}} \tau_{ikc}\Phi_{ikc}(I_i(G_{i,s}(x)))\right\} \\
&= \sum_{x\in\Omega} \log LH(x).
\end{aligned}
$$



2.2 Combined computing

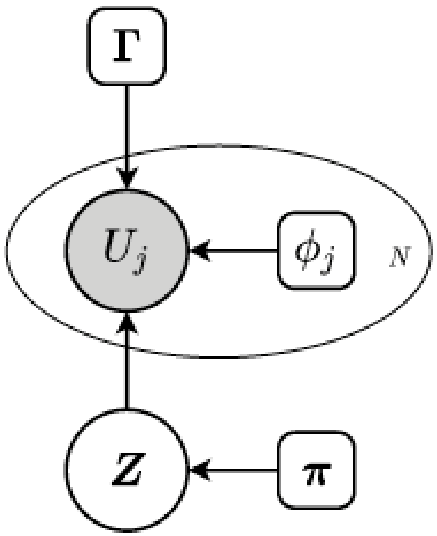
- The parameters of registration and segmentation are updated alternately.
- While X-metric can update segmentation and transformation parameters simultaneously.
- One advantage of combined computing: The registration and segmentation task can benefit each other.

Content

大数据学院
School of Data Science

3.1 Notation and Graphic representation

大数据学院 School of Data Science



(a) Generic framework.

Observed variable:

U_j: The j-th observed image, $\boldsymbol{U} = \{U_j\}_{j=1}^{N}$, $U_j \colon \Omega_\mathrm{j} \to \mathbb{R}$

Latent variable:

Z: Categorical model of the common anatomy

Parameters:

π: Prior proportions of the common anatomy

Γ: Spatial distribution of the common anatomy

ϕ_j: The spatial transformation $\boldsymbol{\phi} = \{\phi_j\}_{j=1}^{N}$, $\phi_j : \Omega \to \Omega_\mathrm{j}$

Images $j = 1, \dots, N$, Common anatomical labels $k = 1, \dots, K$

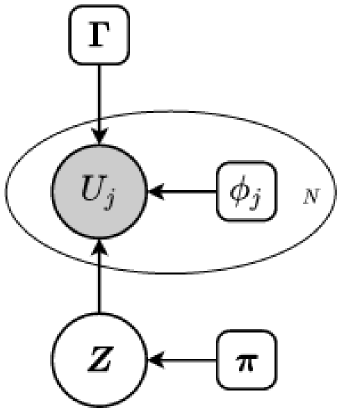
Image space Ω_j , Spatial samples $\boldsymbol{x}, \boldsymbol{\omega}, \boldsymbol{\xi} \in \Omega_j$

Common space / Common coordinate system Ω

$U_j = (u_{j\boldsymbol{\omega}})_{\boldsymbol{\omega} \in \Omega_j}$, where $u_{j\boldsymbol{\omega}} \triangleq U_j \circ \phi_j(\boldsymbol{\omega})$

$\boldsymbol{u}_{\boldsymbol{x}}^{\boldsymbol{\phi}} = [u_{x,1}^{\phi_1}, \dots, u_{x,N}^{\phi_N}]^T$, where $u_{x,j}^{\phi_j} \triangleq U_j \circ \phi_j(\boldsymbol{x})$

3.1 Notation and Graphic representation



(a) Generic framework.

$P(\boldsymbol{U},\boldsymbol{Z};\ \boldsymbol{\phi},\boldsymbol{\pi},\boldsymbol{\Gamma}) = P(\boldsymbol{U}|\boldsymbol{Z};\ \phi,\boldsymbol{\Gamma})\, P(\boldsymbol{Z};\ \boldsymbol{\pi})$

Joint distribution likelihood prior

Inference: posterior $P(\boldsymbol{Z}|\boldsymbol{U};\ \boldsymbol{\phi},\boldsymbol{\pi},\boldsymbol{\Gamma})$

Learning: likelihood $P(\boldsymbol{U}|\boldsymbol{Z};\ \boldsymbol{\phi},\boldsymbol{\Gamma})$

$$
\begin{aligned}
&P(\boldsymbol{U},\boldsymbol{Z};\boldsymbol{\phi},\boldsymbol{\Gamma},\boldsymbol{\pi}) \\
&= P(\boldsymbol{U} \mid \boldsymbol{Z};\boldsymbol{\phi},\boldsymbol{\Gamma}) \cdot P(\boldsymbol{Z};\boldsymbol{\pi}) \\
&= P(\boldsymbol{Z};\boldsymbol{\pi}) \prod_{j=1}^{N} P(U_j \mid \boldsymbol{Z};\phi_j,\boldsymbol{\Gamma}) \\
&= \prod_{\boldsymbol{x}\in\Omega^{\phi}} \prod_{k=1}^{K} \left[P(z_{\boldsymbol{x},k}=1;\boldsymbol{\pi}) \prod_{j=1}^{N} P\Big(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j \mid z_{\boldsymbol{x},k}=1;\phi_j,\boldsymbol{\Gamma}\Big)\right]^{\mathbb{1}(z_{\boldsymbol{x},k}=1)}
\end{aligned}
$$

大数据学院 School of Data Science

Complete-data log-likelihood

$$\begin{aligned}
&\log P(\boldsymbol{U}, \boldsymbol{Z}; \boldsymbol{\phi}, \boldsymbol{\Gamma}, \boldsymbol{\pi}) \\
&= \log \prod_{\boldsymbol{x}\in\Omega^{\phi}} \prod_{k=1}^{K} \left[P(z_{\boldsymbol{x},k} = 1; \boldsymbol{\pi}) \prod_{j=1}^{N} P\left(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j \mid z_{\boldsymbol{x},k} = 1; \phi_j, \boldsymbol{\Gamma}\right)\right]^{\mathbb{1}(z_{\boldsymbol{x},k}=1)} \\
&= \sum_{\boldsymbol{x}\in\Omega^{\phi}} \log \prod_{k=1}^{K} \left[P(z_{\boldsymbol{x},k} = 1; \boldsymbol{\pi}) \prod_{j=1}^{N} P\left(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j \mid z_{\boldsymbol{x},k} = 1; \phi_j, \boldsymbol{\Gamma}\right)\right]^{\mathbb{1}(z_{\boldsymbol{x},k}=1)} \\
&= \sum_{\boldsymbol{x}\in\Omega^{\phi}} \sum_{k=1}^{K} \mathbb{1}(z_{\boldsymbol{x},k} = 1) \log \left[P(z_{\boldsymbol{x},k} = 1; \boldsymbol{\pi}) \prod_{j=1}^{N} P\left(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j \mid z_{\boldsymbol{x},k} = 1; \phi_j, \boldsymbol{\Gamma}\right)\right] \\
&= \sum_{\boldsymbol{x}\in\Omega^{\phi}} \sum_{k=1}^{K} \mathbb{1}(z_{\boldsymbol{x},k} = 1) \left[\log P(z_{\boldsymbol{x},k} = 1; \boldsymbol{\pi}) + \sum_{j=1}^{N} \log P\left(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j \mid z_{\boldsymbol{x},k} = 1; \phi_j, \boldsymbol{\Gamma}\right)\right]
\end{aligned}$$

Appearance model:

$$f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma}) = P\left(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j \mid z_{\boldsymbol{x},k} = 1; \phi_j, \boldsymbol{\Gamma}\right) \triangleq \frac{1}{\mathcal{Z}_{jk}} \sum_{\boldsymbol{x}\in\Omega_S^{\phi}} \beta_3\left(\frac{u_{\boldsymbol{x},j}^{\phi_j} - \mu_j}{h}\right) \cdot \gamma_{\boldsymbol{x},k}$$

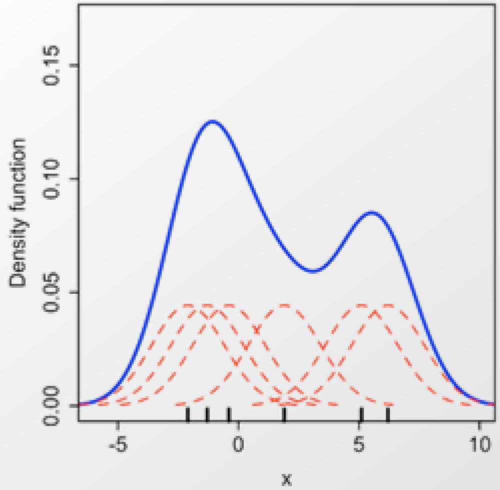
大数据学院 School of Data Science

Kernel density estimation (KDE) is **a non-parametric method** in statistics utilizing kernel smoothing to **estimate the probability density function** of a random variable based on weighted kernels.

Definition: Let $(x_1, x_2, \dots, x_n)$ be independent and identically distributed samples drawn from some univariate distribution with an unknown density f at any given point x. We are interested in estimating the shape of this function f. Its kernel density estimator is

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right),$$

where K is the kernel - a non-negative function, and h is a smoothing parameter called the bandwidth.



Kernel function: $\beta_3(\cdot)$ the cubic B-spline kernel function

Sample weight: $\gamma_{x,k}^{[t]}$

$$f_{jk}^{[t]}\left(\mu_j; \phi_j^{[t]}, \mathbf{\Gamma}^{[t]}\right) = P\left(u_{\boldsymbol{x},j}^{\phi_j^{[t]}} = \mu_j \mid z_{\boldsymbol{x},k} = 1; \phi_j^{[t]}, \mathbf{\Gamma}^{[t]}\right) \triangleq \frac{1}{\mathcal{Z}_{jk}} \sum_{\boldsymbol{x} \in \Omega_S^{\phi^{[t]}}} \beta_3\left(\frac{u_{\boldsymbol{x},j}^{\phi_j^{[t]}} - \mu_j}{h}\right) \cdot \gamma_{\boldsymbol{x},k}^{[t]}$$

https://en.wikipedia.org/wiki/Kernel_density_estimation

3.2 MLE insights and EM

大数据学院 School of Data Science

Expected Complete-data log-likelihood at the t-th step

$$\mathcal{Q}\left(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{[t]}\right) \triangleq \mathbb{E}\left[\ln P(\boldsymbol{U}, \boldsymbol{Z}; \boldsymbol{\theta}) \mid \boldsymbol{U}; \boldsymbol{\theta}^{[t]}\right]$$

$$= \sum_{\boldsymbol{x} \in \Omega^{\phi}} \sum_{k=1}^{K} E(\mathbb{1}(z_{\boldsymbol{x},k} = 1)) \left[\log \pi_k + \sum_{j=1}^{N} \log f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma})\right]$$

$$E(\mathbb{1}(z_{\boldsymbol{x},k} = 1)) = P\left(z_{\boldsymbol{x},k} = 1 \mid \boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}; \boldsymbol{\theta}^{[t]}\right) \triangleq q_{\boldsymbol{x},k}^{[t]}$$

$$q_{\boldsymbol{x},k}^{[t]} = \frac{P\left(z_{\boldsymbol{x},k} = 1, \boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}; \boldsymbol{\theta}^{[t]}\right)}{P\left(\boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}; \boldsymbol{\theta}^{[t]}\right)}$$

$$= \frac{\pi_k^{[t]} \prod_{j=1}^{N} f_{jk}^{[t]}\left(\mu_j; \phi_j^{[t]}, \boldsymbol{\Gamma}^{[t]}\right)}{\sum_{l=1}^{K} \pi_l^{[t]} \prod_{j=1}^{N} f_{jl}^{[t]}\left(\mu_j; \phi_j^{[t]}, \boldsymbol{\Gamma}^{[t]}\right)}$$

Parameters to solved:

Common space parameters: $\boldsymbol{\pi}$, $\boldsymbol{\Gamma}$

Transformation parameters: $\boldsymbol{\phi} = \{\boldsymbol{\phi}_j\}_{j=1}^{N}$

$$=> \mathcal{Q}\left(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{[t]}\right) = \sum_{\boldsymbol{x} \in \Omega^{\phi}} \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \left[\log \pi_k + \sum_{j=1}^{N} \log f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma})\right]$$

π Lagrange multiplier

$$\boldsymbol{\pi}^{[\boldsymbol{t}+\mathbf{1}]} = \arg\max_{\boldsymbol{\pi}} \sum_{\boldsymbol{x}\in\Omega^{\phi}} \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \log \pi_k, \quad \text{s.t.} \sum_{k=1}^{K} \pi_k = 1$$

$$L(\pi, \lambda) = \sum_{x\in\Omega^{\phi}} \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \log \pi_k + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

$$\text{s.t.} \sum_{k=1}^{K} \pi_k = 1$$

$$\Rightarrow \quad \pi_k = \frac{\sum_{\boldsymbol{x}\in\Omega^{\phi}} q_{\boldsymbol{x},k}^{[t]}}{\sum_{k=1}^{K} \sum_{\boldsymbol{x}\in\Omega^{\phi}} q_{\boldsymbol{x},k}^{[t]}}$$

大数据学院 School of Data Science

Γ spatial distribution

$$f_{jk}(\mu_j;\phi_j,\boldsymbol{\Gamma}) = \frac{1}{\mathcal{Z}_{jk}} \sum_{\boldsymbol{x}\in\Omega_S^{\phi}} \beta_3\left(\frac{u_{\boldsymbol{x},j}^{\phi_j}-\mu_j}{h}\right)\cdot\gamma_{\boldsymbol{x},k}$$

$$\boldsymbol{\Gamma}^{[t+1]} = \arg\max_{\boldsymbol{\Gamma}} \sum_{\boldsymbol{\omega}\in\Omega^{\phi}} \sum_{k=1}^{K} q_{\boldsymbol{\omega},k}^{[t]} \sum_{j=1}^{N} \log \sum_{\boldsymbol{\xi}\in\Omega^{\phi}} \beta_3\left(\frac{u_{\boldsymbol{\xi},j}^{\phi_j}-\mu_j}{h}\right)\cdot\gamma_{\boldsymbol{\xi},k}$$

$$\text{s.t. } \sum_{k=1}^{k}\gamma_{\boldsymbol{x},k} = 1, \forall \boldsymbol{x}\in\Omega^{\phi}$$

$$\Longleftrightarrow \quad \gamma_{\boldsymbol{x}}^{[t+1]} = \arg\max_{\gamma_{\boldsymbol{x}}} \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log(a_j\cdot\gamma_{\boldsymbol{x},k}+b_j)$$

$$\text{s.t. } \sum_{k=1}^{k}\gamma_{\boldsymbol{x},k} = 1,$$

$$\text{where}\begin{cases} a_j \triangleq \beta_3\left(\frac{u_{\boldsymbol{x},j}^{\phi_j}-\mu_j}{h}\right) \\ b_j \triangleq \sum_{\boldsymbol{\xi}\in\Omega^{\phi}\setminus\{\boldsymbol{x}\}}\beta_3\left(\frac{u_{\boldsymbol{\xi},j}^{\phi_j}-\mu_j}{h}\right)\cdot\gamma_{\boldsymbol{\xi},k} \end{cases}, j=1,\ldots,N$$

$$\text{and} \quad a_j, b_j \geq 0$$

大数据学院 School of Data Science

Γ spatial distribution

$$H(\gamma_{\boldsymbol{x}}) = \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log(a_j \cdot \gamma_{\boldsymbol{x},k} + b_j)$$

$$\sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log(a_j \cdot \gamma_{\boldsymbol{x},k} + b_j) \geqslant$$

$$\sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log a_j \gamma_{\boldsymbol{x},k} = N \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \log \gamma_{\boldsymbol{x},k} + \sum_{j=1}^{N} \log a_j \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]}$$

$$\gamma_{\boldsymbol{x}}^{[t+1]} = \arg\max_{\gamma_{\boldsymbol{x}}} N \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \log \gamma_{\boldsymbol{x},k} + \sum_{j=1}^{N} \log a_j$$

$$= \arg\max_{\gamma_{\boldsymbol{x}}} \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \log \gamma_{\boldsymbol{x},k}$$

$$\text{s.t.} \sum_{k=1}^{k} \gamma_{\boldsymbol{x},k} = 1,$$

$$\Rightarrow \quad \gamma_{\boldsymbol{x},k}^{[t+1]} = q_{\boldsymbol{x},k}^{[t]}$$

大数据学院 School of Data Science

Γ spatial distribution

Demonstrate the rationality:

$$\gamma_{\boldsymbol{x},k}^{[t+1]} = q_{\boldsymbol{x},k}^{[t]}, \quad k = 1, \ldots, K,\ t \geq 0, \quad s.t.\ H(\gamma_{\boldsymbol{x}}^{[t+1]}) \geq H(\gamma_{\boldsymbol{x}}^{[t]})$$

Proof:

$$H\left(\gamma_{\boldsymbol{x}}^{[t+1]}\right) - H\left(\gamma_{\boldsymbol{x}}^{[t]}\right) = \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log \frac{a_j \gamma_{\boldsymbol{x},k}^{[t+1]} + b_j}{a_j \gamma_{\boldsymbol{x},k}^{[t]} + b_j} = \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log \frac{a_j q_{\boldsymbol{x},k}^{[t]} + b_j}{a_j q_{\boldsymbol{x},k}^{[t-1]} + b_j}$$

(1). If $q_{xk}^{[t]} \leqslant q_{xk}^{[t-1]}$

$$\frac{a_j q_{\boldsymbol{x},k}^{[t]} + b_j}{a_j q_{\boldsymbol{x},k}^{[t-1]} + b_j} \geqslant \frac{q_{\boldsymbol{x},k}^{[t]}}{q_{\boldsymbol{x},k}^{[t-1]}}, \quad a_j, b_j \geqslant 0$$

$$H\left(\gamma_{\boldsymbol{x}}^{[t+1]}\right) - H\left(\gamma_{\boldsymbol{x}}^{[t]}\right) \geqslant \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log \frac{q_{\boldsymbol{x},k}^{[t]}}{q_{\boldsymbol{x},k}^{[t-1]}} = N \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \log \frac{q_{\boldsymbol{x},k}^{[t]}}{q_{\boldsymbol{x},k}^{[t-1]}} \geqslant 0$$

Γ spatial distribution

Demonstrate the rationality:

$$\gamma_{\boldsymbol{x},k}^{[t+1]} = q_{\boldsymbol{x},k}^{[t]}, \quad k = 1, \ldots, K,\ t \geq 0, \quad s.t.\ H(\gamma_{\boldsymbol{x}}^{[t+1]}) \geq H(\gamma_{\boldsymbol{x}}^{[t]})$$

Proof:

$$H\left(\gamma_{\boldsymbol{x}}^{[t+1]}\right) - H\left(\gamma_{\boldsymbol{x}}^{[t]}\right) = \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log \frac{a_j \gamma_{\boldsymbol{x},k}^{[t+1]} + b_j}{a_j \gamma_{\boldsymbol{x},k}^{[t]} + b_j} = \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log \frac{a_j q_{\boldsymbol{x},k}^{[t]} + b_j}{a_j q_{\boldsymbol{x},k}^{[t-1]} + b_j}$$

(2). Else if $q_{xk}^{[t]} > q_{xk}^{[t-1]}$

We have known that $1 \geqslant q_{xk}^{[t]} > q_{xk}^{[t-1]} \geqslant 0$,

$$\Rightarrow \frac{a_j q_{\boldsymbol{x},k}^{[t]} + b_j}{a_j q_{\boldsymbol{x},k}^{[t-1]} + b_j} > 1$$

$$H\left(\gamma_{\boldsymbol{x}}^{[t+1]}\right) - H\left(\gamma_{\boldsymbol{x}}^{[t]}\right) \geqslant \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log 1 = 0 \qquad \square$$

Set

$$\gamma_{\boldsymbol{x},k}^{[t+1]} = q_{\boldsymbol{x},k}^{[t]} = \frac{\pi_k^{[t]} \prod_{j=1}^{N} f_{jk}^{[t]}\left(\mu_j; \phi_j^{[t]}, \boldsymbol{\Gamma}^{[t]}\right)}{\sum_{l=1}^{K} \pi_l^{[t]} \prod_{j=1}^{N} f_{jl}^{[t]}\left(\mu_j; \phi_j^{[t]}, \boldsymbol{\Gamma}^{[t]}\right)}$$

$$\pi_k^{[t+1]} = \frac{\sum_{\boldsymbol{x}\in\Omega^{\phi^{[t]}}} q_{\boldsymbol{x},k}^{[t]}}{\sum_{k=1}^{K} \sum_{\boldsymbol{x}\in\Omega^{\phi^{[t]}}} q_{\boldsymbol{x},k}^{[t]}} = \frac{\sum_{\boldsymbol{x}\in\Omega^{\phi^{[t]}}} \gamma_{\boldsymbol{x},k}^{[t+1]}}{\sum_{k=1}^{K} \sum_{\boldsymbol{x}\in\Omega^{\phi^{[t]}}} \gamma_{\boldsymbol{x},k}^{[t+1]}}$$

3.2 MLE insights and EM

ϕ *transformation*

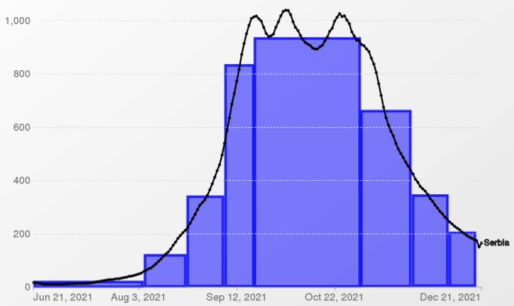
$$\mathcal{S}\left(\boldsymbol{\phi} \mid \boldsymbol{\theta}^{[t]}\right) \triangleq \sum_{\boldsymbol{x} \in \Omega^{\phi}} \sum_{k=1}^{K} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma})$$

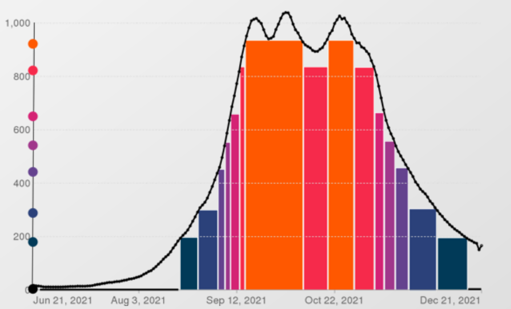
$$q_{\boldsymbol{x},k}^{[t]} = P\left(z_{\boldsymbol{x},k} = 1 \mid \boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}; \boldsymbol{\theta}^{[t]}\right) \qquad P\left(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j \mid z_{\boldsymbol{x},k} = 1; \phi_j, \boldsymbol{\Gamma}\right)$$

$$P(\boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}) \quad ?$$

$$\boldsymbol{\Omega}_{\boldsymbol{\mu}}^{\phi} \triangleq \left\{\boldsymbol{x} \in \boldsymbol{\Omega}^{\phi} \mid \boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}\right\}$$

$$q_{\boldsymbol{x},k} = q_{\boldsymbol{\mu},k}, \quad \forall \boldsymbol{x} \in \boldsymbol{\Omega}_{\boldsymbol{\mu}}^{\phi}$$





Riemannian (top) vs Lebesgue (bottom) integration =>

3.2 MLE insights and EM

大数据学院 School of Data Science

ϕ *transformation*

$$\mathcal{S}\left(\phi \mid \boldsymbol{\theta}^{[t]}\right) = \sum_{k=1}^{K} \sum_{\mu} \sum_{\boldsymbol{x} \in \boldsymbol{\Omega}_{\mu}^{\phi}} q_{\boldsymbol{x},k}^{[t]} \sum_{j=1}^{N} \log f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma})$$

$$= \sum_{k=1}^{K} \sum_{\mu} |\boldsymbol{\Omega}_{\boldsymbol{\mu}}^{\phi}| q_{\boldsymbol{\mu},k}^{[t]} \sum_{j=1}^{N} \log f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma})$$

$$\frac{\left|\boldsymbol{\Omega}_{\boldsymbol{\mu}}^{\phi}\right|}{\left|\boldsymbol{\Omega}^{\phi}\right|} \xrightarrow{\text{a.s.}} P^*\left(\boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}\right), \quad \text{as } \left|\boldsymbol{\Omega}^{\phi}\right| \to \infty$$

$$P^*\left(\boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}\right) \approx P\left(\boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}\right), \quad \forall \boldsymbol{x} \in \boldsymbol{\Omega}^{\phi}$$

$$\left|\boldsymbol{\Omega}_{\boldsymbol{\mu}}^{\phi}\right| \approx \left|\boldsymbol{\Omega}^{\phi}\right| P\left(\boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}\right)$$

3.2 MLE insights and EM

$$
\begin{aligned}
\mathcal{S}\left(\boldsymbol{\phi} \mid \boldsymbol{\theta}^{[t]}\right) &= \sum_{k=1}^{K} \sum_{\boldsymbol{\mu}} |\boldsymbol{\Omega}_{\boldsymbol{\mu}}^{\phi}| q_{\boldsymbol{\mu},k}^{[t]} \sum_{j=1}^{N} \log f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma}) \\
&\approx \sum_{k=1}^{K} \sum_{\boldsymbol{\mu}} |\boldsymbol{\Omega}^{\phi}| P(\boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}) q_{\boldsymbol{\mu},k}^{[t]} \sum_{j=1}^{N} \log f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma}) \\
&= \sum_{k=1}^{K} \sum_{\boldsymbol{\mu}} |\boldsymbol{\Omega}^{\phi}| P\left(z_{\boldsymbol{x},k} = 1, \boldsymbol{u}_{\boldsymbol{x}}^{\phi} = \boldsymbol{\mu}; \boldsymbol{\theta}^{[t]}\right) \sum_{j=1}^{N} \log f_{jk}(\mu_j; \phi_j, \boldsymbol{\Gamma}) \\
&= -|\boldsymbol{\Omega}^{\phi}| \sum_{j=1}^{N} H(U_j \circ \phi_j | \boldsymbol{Z}; \boldsymbol{\Gamma})
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{L}\left(\boldsymbol{\phi} \mid \boldsymbol{U}; \boldsymbol{\Gamma}^{[t+1]}\right) &\triangleq -\mathcal{X}\left(\boldsymbol{U}[\boldsymbol{\phi}], \boldsymbol{Z}^{[t+1]}\right) + \lambda \cdot R(\boldsymbol{\phi}) \\
&= -\sum_{j=1}^{N} I\left(U_j \circ \phi_j, \boldsymbol{Z}^{[t+1]}\right) + \lambda \cdot R(\boldsymbol{\phi})
\end{aligned}
$$

$$
\Rightarrow \quad \boldsymbol{\phi}^{[t+1]} = \boldsymbol{\phi}^{[t]} - \eta \cdot \nabla \mathcal{L}\left(\boldsymbol{\phi} \mid \boldsymbol{U}; \boldsymbol{\Gamma}^{[t+1]}\right)\Big|_{\boldsymbol{\phi}=\boldsymbol{\phi}^{[t]}}
$$

3.2 MLE insights and EM

大数据学院
School of Data Science

$\mathcal{X}$-metric

A combination of total correlation and intensity-class mutual information

$$C(\boldsymbol{U}) \triangleq D_{\mathrm{KL}}\left[P(\boldsymbol{U}) \| \prod_{j=1}^{N} P(U_j)\right] = \left[\sum_{j=1}^{N} H(U_j)\right] - \boxed{H(\boldsymbol{U})}$$

$$I(\boldsymbol{U}, \boldsymbol{Z}) = H(\boldsymbol{U}) - H(\boldsymbol{U} \mid \boldsymbol{Z}) = \boxed{H(\boldsymbol{U})} - \sum_{j=1}^{N} H(U_j \mid \boldsymbol{Z})$$

$$\mathcal{X}(\boldsymbol{U}, \boldsymbol{Z}) = \sum_{j=1}^{N} I(U_j, \boldsymbol{Z}) = \sum_{j=1}^{N} [H(U_j) + H(\boldsymbol{Z}) - H(U_j, \boldsymbol{Z})]$$
$$= \sum_{j=1}^{N} [H(\boldsymbol{Z}) - H(\boldsymbol{Z} \mid U_j)]$$

- The reduction in uncertainty of the common anatomy due to the observation.
- The sharpening of the inferred common anatomy. That is, as $I(U, Z)$ increases, the conditional entropy would $\mathrm{H}(Z|\,U)$ reduce and the posterior distribution $\mathrm{P}(Z|\,U)$ would become more concentrated.

3.2 MLE insights and EM

大数据学院 School of Data Science

$\mathcal{X}$-metric

$$I\left(U_j \circ \phi_j, \boldsymbol{Z}^{[t+1]}\right) = \sum_{\mu_j} \sum_{k=1}^{K} p_j^{[t+1]}(\mu_j, k; \phi_j) \log \frac{p_j^{[t+1]}(\mu_j, k; \phi_j)}{p_j^{[t+1]}(\mu_j; \phi_j) p_j^{[t+1]}(k; \phi_j)}$$

$$p_j^{[t+1]}(\mu_j, k; \phi_j) \triangleq P\left(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j, z_{\boldsymbol{x},k} = 1; \boldsymbol{\Gamma}^{[t+1]}\right)$$

$$p_j^{[t+1]}(\mu_j; \phi_j) \triangleq \sum_{k=1}^{K} p_j^{[t+1]}(\mu_j, k; \phi_j)$$

$$p_j^{[t+1]}(k; \phi_j) \triangleq \sum_{\mu_j} p_j^{[t+1]}(\mu_j, k; \phi_j)$$

Kernel density estimator (KDE)

$$p_j^{[t+1]}(\mu_j, k; \phi_j) \triangleq \frac{1}{\mathcal{Z}_j} \sum_{x \in \Omega_{S'}^{\phi}} \beta_3\left(\frac{u_{\boldsymbol{x},j}^{\phi_j} - \mu_j}{h}\right) \cdot \gamma_{\boldsymbol{x},k}^{[t+1]}$$

3.2 MLE insights and EM

大数据学院 School of Data Science

MLE ⇨ $$\widehat{\phi} = \arg\max_{\phi} \max_{\alpha} \mathcal{X}(\boldsymbol{U}[\phi], \boldsymbol{Z})$$

Algorithm 1. $\mathcal{X}$-CoReg

Data: The observed images $U = \{U_j\}_{j=1}^{N}$;
Input: Number of iterations T, regularization coefficient λ, registration step size η;
Output: The estimated spatial transformations $\widehat{\boldsymbol{\phi}} = \{\widehat{\phi}_j\}_{j=1}^{N}$;

```
Initialization: φ_j^[0] ≜ id for j = 1, ..., N; initialize π^[0] and Γ^[0] by (22);
for t = 0, ..., T − 1 do
    /* Update the common-space parameters */
    γ_{x,k}^[t+1] ≜ π_k^[t] ∏_{j=1}^N f_jk^[t](μ_j; φ_j^[t]) / Σ_{k=1}^K π_k^[t] ∏_{j=1}^N f_jk^[t](μ_j; φ_j^[t]);
    π_k^[t+1] ≜ Σ_{x∈Ω^{φ[t]}} γ_{x,k}^[t+1] / Σ_{k=1}^K Σ_{x∈Ω^{φ[t]}} γ_{x,k}^[t+1];
    /* Update the spatial transformations */
    φ^[t+1] = φ^[t] − η · ∇L(φ | U; Γ^[t+1]) |_{φ=φ^[t]};
    if L converges then
        break loop;
return φ̂ = φ^[T].
```

Common space parameters: $\boldsymbol{\pi}$, $\boldsymbol{\Gamma}$

Transformation parameters: $\boldsymbol{\phi} = \{\boldsymbol{\phi}_j\}_{j=1}^{N}$

$$\gamma_{\boldsymbol{x},k}^{[t+1]} = \frac{\pi_k^{[t]} \prod_{j=1}^{N} f_{jk}^{[t]}\left(\mu_j; \phi_j^{[t]}, \boldsymbol{\Gamma}^{[t]}\right)}{\sum_{l=1}^{K} \pi_l^{[t]} \prod_{j=1}^{N} f_{jl}^{[t]}\left(\mu_j; \phi_j^{[t]}, \boldsymbol{\Gamma}^{[t]}\right)}$$

$$\pi_k^{[t+1]} = \frac{\sum_{\boldsymbol{x}\in\Omega^{\phi(t)}} \gamma_{\boldsymbol{x},k}^{[t+1]}}{\sum_{k=1}^{K} \sum_{\boldsymbol{x}\in\Omega^{\phi(t)}} \gamma_{\boldsymbol{x},k}^{[t+1]}}$$

$$\boldsymbol{\phi}^{[t+1]} = \boldsymbol{\phi}^{[t]} - \eta \cdot \nabla\mathcal{L}\left(\boldsymbol{\phi} \mid \boldsymbol{U}; \boldsymbol{\Gamma}^{[t+1]}\right)\Big|_{\boldsymbol{\phi}=\boldsymbol{\phi}^{[t]}}$$

Content

大数据学院
School of Data Science

4.1 Framework Modification

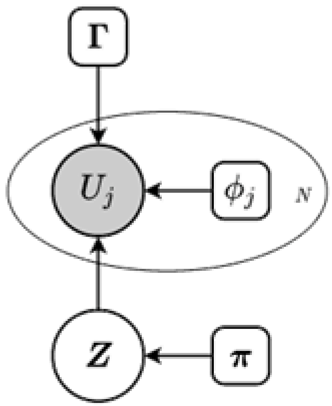
大数据学院
School of Data Science

- Introduce of segmentation masks of partial images
  - $\{Y_j\}_{j \in \mathcal{J}}$, the available segmentation masks of the corresponding observed images $\{U_j\}_{j \in \mathcal{J}}$, where $\mathcal{J}$ is an index set.
    - Y_j is a categorical random field
      - $Y_j = (y_{j\omega})_{\omega \epsilon \Omega_j}$
      - $y_{j\omega} = [y_{j\omega,1}, \ldots, y_{j\omega,K}] \in \{0,1\}^K$, a one-hot vector
  - ρ_j, the probability maps of the segmentation Y_j

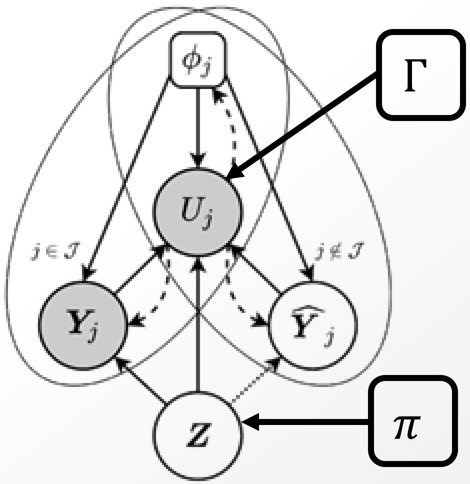
$$\rho_{jk}(\phi_j(\boldsymbol{x})) = \begin{cases} P\left(y_{j\phi_j(\boldsymbol{x}),k} = 1 \mid z_{\boldsymbol{x},k} = 1\right) = \tilde{\rho}_{j\phi_j(\boldsymbol{x}),k} \propto \exp[\tau \cdot D_{jk}(\phi_j(\boldsymbol{x}))],\ j \in \mathcal{J} & \text{有GT, observed label, 模糊化} \\ P\left(\hat{y}_{j\phi_j(\boldsymbol{x}),k} = 1 \mid U_j\right) = \hat{\rho}_{j\phi_j(\boldsymbol{x}),k},\ j \notin \mathcal{J} & \text{无GT, probability map predicted from network} \end{cases}$$

where D_{jk} is the signed distance map of Y_j for label k, and τ controls the slope of the distance.

Generic Framework & Extended Framework



(a) Generic framework.



(b) Extended framework.

- Circles: Random variables
- Rounded boxes: Deterministic parameters
- Shaded circles: Observed variables
- Ellipses: Replication
- Solid arrows: Generation
- Dashed arrows: Inference procedure from a neural network
- Dotted arrows indicate that the corresponding conditional probability distribution is not incorporated in posterior computation

- U_j: the j-th observed image
- Z: categorical model of the common anatomy
- π: prior proportions of the common anatomy
- Γ: spatial distribution of the common anatomy
- ϕ_j: the spatial transformation from Ω to Ω_j
- Y_j: the available segmentation mask of U_j
- $\hat{Y}_j$: the predicted segmentation mask of U_j

4.2 MLE => Loss function

大数据学院
School of Data Science

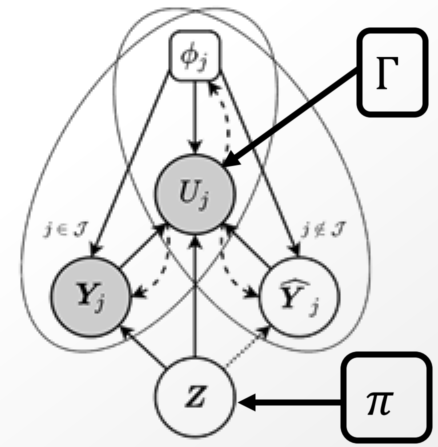
Compute Q function

$$P(U,Z,Y;\ \phi,\Gamma,\pi)$$

$$= P(U|Y,Z;\ \phi,\Gamma)P(Y|Z;\phi)P(Z;\pi)$$

$$= P(Z;\pi)\prod_{j=1}^{N} P\big(u_j\big|\ y_j,Z;\ \phi_j,\Gamma\big)P\big(y_j\big|Z,\phi_j\big)$$

$$= \prod_{x\,\epsilon\,\Omega^{\phi}}\prod_{k=1}^{K}\Big[P(z_{x,k}=1;\ \pi)\prod_{j=1}^{N}P(u_{jx}^{\phi_j}|y_{j\phi_j(x),k}=1, z_{x,k}=1;\ \phi_j,\Gamma)\ P\Big(y_{j\phi_j(x),k}=1\Big|\ z_{x,k}=1,\phi_j\Big)\Big]^{1(z_{x,k}=1)}$$



(b) Extended framework.

$$logP(U,Z,Y;\ \phi,\Gamma,\pi)$$

$$= \sum_{x\,\in\,\Omega^{\phi}}\sum_{k=1}^{K} 1\big(z_{x,k}=1\big)\Big[log\gamma_{x,k} + \sum_{j=1}^{N} logf_{jk}\big(\mu_j;\ \phi_j,\Gamma\big)P(y_{j\phi_j(x),k}=1|z_{x,k}=1,\phi_j)\Big]$$

关于Z$|U,Y;\ \theta^{[t]}$求期望，可得

$$Q(\theta|\theta^{[t]}) = \sum_{x\epsilon\Omega^{\phi}}\sum_{k=1}^{K} q_{x,k}^{[t]}\Big[\log\gamma_{x,k} + \sum_{j=1}^{N}\log f_{jk}\big(\mu_j;\ \phi_j,\Gamma\big)P(y_{j\phi_j(x),k}=1|z_{x,k}=1,\phi_j)\Big]$$

4.2 MLE => Loss function

大数据学院 School of Data Science

- 按照前面类似的求解方法，可得

$$\gamma_{x,k}^{[t+1]} = q_{x,k}^{[t]} \qquad \pi_k^{[t+1]} = \frac{\sum_{x \in \Omega^\phi} \gamma_{x,k}^{[t+1]}}{\sum_{k=1}^{K} \sum_{x \in \Omega^\phi} \gamma_{x,k}^{[t+1]}}$$

- 给出$q_{x,k}^{[t]}$的估计式

  - $\mathrm{q}_{x,k} = P\left(z_{x,k} = 1 \middle| u_x^\phi, y_x^\phi;\ \phi, \Gamma, \pi\right)$

$$\propto P(z_{x,k} = 1;\ \pi) \prod_{j=1}^{N} P\left(y_{j\phi_j(x),k=1 |\, z_{x,k}=1;\ \phi_j}\right) P(u_{x,j}^{\phi_j} = \mu_j | y_{j\phi_j(x),k} = 1, z_{x,k} = 1;\ \phi_j, \Gamma) \quad (1)$$

$$\approx P(z_{\boldsymbol{x},k} = 1; \boldsymbol{\pi}) \prod_{j=1}^{N} P(u_{\boldsymbol{x},j}^{\phi_j} = \mu_j \mid y_{j\phi_j(\boldsymbol{x}),k} = 1, z_{\boldsymbol{x},k} = 1; \phi_j, \boldsymbol{\Gamma}) \quad (2)$$

$$\approx \pi_k^{[t]} \prod_{j=1}^{N} f_{jk}^{[t]}(\mu_j), \quad \text{for } k = 1, \ldots, K \text{ and } \forall x \in \Omega. \quad (3)$$

其中，$f_{jk}^{[0]}(\mu_j) \propto \sum_{\boldsymbol{\omega} \in \Omega_j} \beta_3\left(\frac{U_j(\boldsymbol{\omega}) - \mu_j}{h}\right) \cdot \rho_{jk}(\boldsymbol{\omega})$

$$f_{jk}^{[t]}(\mu_j) \propto \sum_{\boldsymbol{x} \in \Omega_S^{\phi_j^{[t]}}} \beta_3\left(\frac{u_{\boldsymbol{x},j}^{\phi_j^{[t]}} - \mu_j}{h}\right) \cdot \gamma_{\boldsymbol{x},k}^{[t]}, \quad t \geq 1$$

4.2 MLE => Loss function

$$logP(U,Z,Y;\ \phi,\Gamma,\pi)$$

$$= \sum_{x\in\Omega^{\phi}}\sum_{k=1}^{K}1(z_{x,k}=1)[log\gamma_{x,k}+\sum_{j=1}^{N}logf_{jk}(\mu_j;\ \phi_j,\Gamma)P(y_{j\phi_j(x),k}=1|z_{x,k}=1,\phi_j)]$$

$$S(\phi,\Gamma;\ \theta^{[t]}) = \sum_{x\in\Omega^{\phi}}[\underbrace{\sum_{k=1}^{K}q_{x,k}^{[t]}\sum_{j=1}^{N}\log f_{jk}(\mu_j;\ \phi_j,\Gamma)}_{(1)}+\underbrace{\sum_{k=1}^{K}q_{x,k}^{[t]}\sum_{j=1}^{N}\log P\left(y_{j\phi_{j(x)},k}=1\middle|\,z_{x,k}=1,\phi_j\right)]}_{(2)}$$

(1): the cross entropy between the posterior and the appearance model, approximate it using the proposed X-metric

$$L_1(\phi)\ \triangleq -\chi\left(U[\phi],Z^{[2]}\right)+\ \lambda\cdot R(\phi)$$

(2): the cross entropy between the posterior and the warped probability maps → a hybrid loss

$$L_2(\phi,\hat{\rho})\ \triangleq \sum_{j\,\epsilon\,\mathcal{J}}H_{Z^{[2]}}\left(Y_j\circ\ \phi_j\right)+\sum_{j\,\notin\,\mathcal{J}}H_{Z^{[2]}}\left(\hat{Y}_j\circ\ \phi_j\right)$$

where $H_{Z^{[2]}}\left(Y_j\circ\phi_j\right)\triangleq -\frac{1}{|\Omega|}\sum_{x\in\Omega}\sum_{k=1}^{K}\gamma_{x,k}^{[2]}\log\rho_{jk}(\phi_j(x))$,

$H_{Z^{[2]}}(\hat{Y}_j\circ\phi_j)\triangleq -\frac{1}{|\Omega|}\sum_{x\in\Omega}\sum_{k=1}^{K}\gamma_{x,k}^{[2]}\log\hat{\rho}_{jk}(\phi_j(x))$.

4.2 EM => Loss function

- Finally, a segmentation loss is included, i.e.,

$$L_3(\hat{\rho}) \triangleq -\underbrace{\sum_{j=1}^{N} I\left(U_j, \hat{Y}_j\right)}_{} + \underbrace{\sum_{j \notin \mathcal{J}} H\left(\hat{Y}_j\right)}_{} + \underbrace{\sum_{j \in \mathcal{J}} L_{seg}(Y_j, \hat{Y}_j)}_{}$$

(1): optimize probability maps based on image intensities

(2): encourage the probability vector $[\hat{\rho}_{j1}(w), \ldots, \hat{\rho}_{jK}(w)]$ to be concentrated

(3): $L_{seg}(Y_j, \hat{Y}_j) \triangleq H_{Y_j}(\hat{Y}_j)$ + [1 - DSC($Y_j, \hat{Y}_j$)], measure the discrepancy between the network prediction and the ground-truth segmentation

- The total loss function: $L(\phi, \hat{\rho}) \triangleq L_1(\phi) + L_2(\phi, \hat{\rho}) + L_3(\hat{\rho})$

4.3 Network Architecture

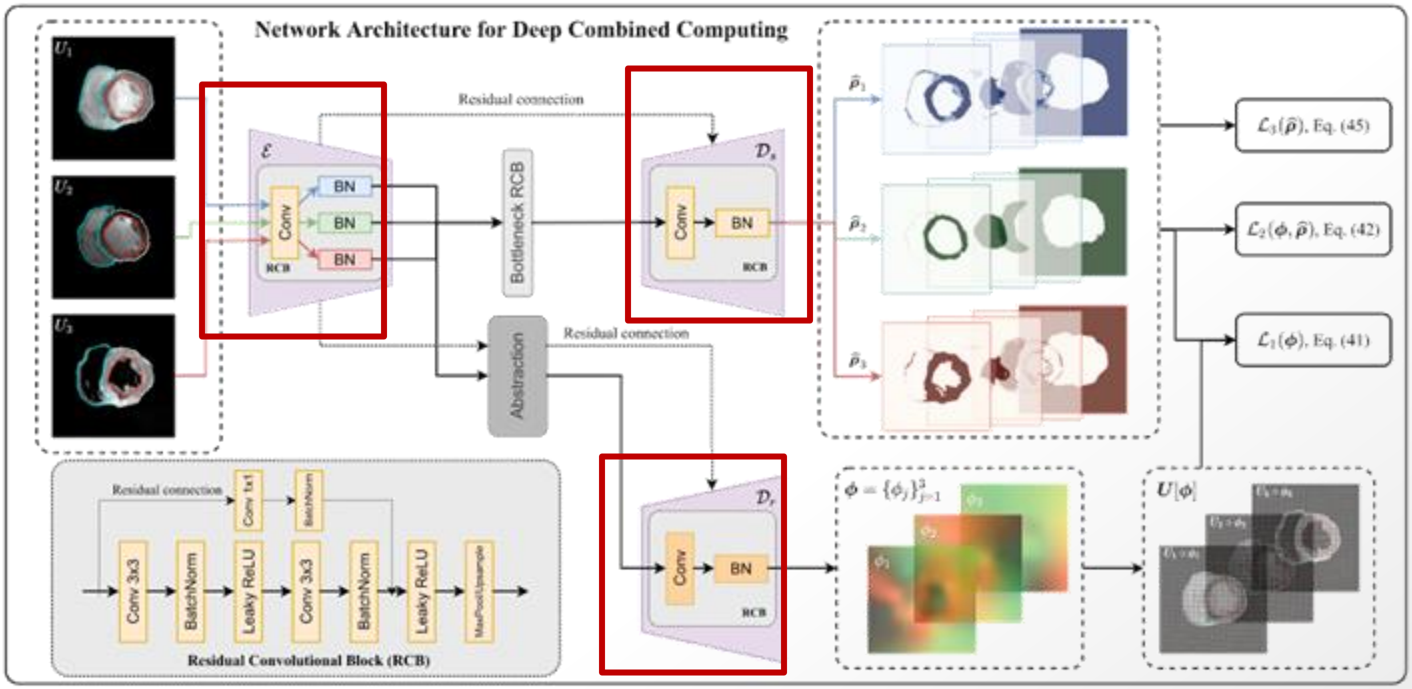


Fig. 5. An example of the network architecture for deep combined computing when $N = 3$. The encoder $\mathcal{E}$, the decoders $\mathcal{D}_s$ and $\mathcal{D}_r$, and the bottleneck are composed of residual convolutional blocks. Domain-specific batch normalization layers are indicated with different colours. Cardiac structures, *i.e.* myocardium, left ventricle (LV) and right ventricle (RV), are rendered as contours.

- The network is composed of an encoder $\mathcal{E}$, a bottleneck, a segmentation decoder D_s and a registration decoder D_r.
- They comprise multiple levels of residual convolutional blocks (RCBs) and residual connections between the encoder and decoder.

4.3 Training Pipeline

- The network parameters for segmentation and registration are optimized alternately so that the improvement of one task can benefit the other.
- We choose to alternate training between the two branches because in Eq.(42) the term $H_{Z^{[2]}}(\hat{Y}_j \circ \phi_j)$ is computed using both branches.
    - $\hat{Y}_j$ is predicted by the segmentation branch.
    - ϕ_j is predicted by the registration branch.
    - To avoid interference in two branches, like the situation where the registration branch may seek to compensate for errors in the segmentation prediction, it could be better to alternate the training for the two branches.

Registration: L_1, L_2
Update ϕ

Segmentation: L_2, L_3
Update $\hat{\rho}$

Content

Experiment 1: Group Registration

- Multimodal nonrigid groupwise registration for multi-sequence brain MRI
- The **BrainWeb** online database1 provides simulated **T1-, T2- and PD-weighted** MRI volumes from **an anatomical phantom**
- spacing of 181×217×181 mm3
- K = 4: cerebrospinal fluid (CSF), grey matter (GM), and white matter (WM), background
- Generate the initial misalignments
- multi-level isotropic FFDs
- deformation regularization was imposed by bending energy over the FFD meshes, with λ=0.001
- Adam optimizer, initial step size η=0.1

Code: Main Pipeline

大数据学院
School of Data Science

```
train_provider = ImageDataProvider(dimension=2, data_search_path=args.test_data_search_path, a_min=args

net = BrainWebRegModel(dimension=2, ⋯

trainer = IterGroupRegTrainer(net, verbose=0, save_path=save_path,
                              optimizer_name=args.optimizer,
                              learning_rate=args.learning_rate,
                              weight_decay=args.weight_decay,
                              scheduler_name=args.scheduler,
                              base_lr=args.base_learning_rate,
                              max_lr=args.max_learning_rate,
                              logger=logger)

phantom = image_utils.load_image_nii(os.path.join(args.test_data_search_path,
                                                  'redefined_phantom_1.0mm_normal_crisp.nii.gz'))[0]
phantom = phantom[phantom.shape[0] // 2]
phantom = image_utils.get_one_hot_label(phantom, label_intensities=(0, 1, 2, 3), channel_first=True)
phantom = torch.from_numpy(phantom).unsqueeze(0)

indices, pre_metrics, post_metrics = trainer.train(train_provider, phantom=phantom, device=device,
                                                   steps=args.steps, display_step=args.display_step,
                                                   num_workers=args.num_workers)
```

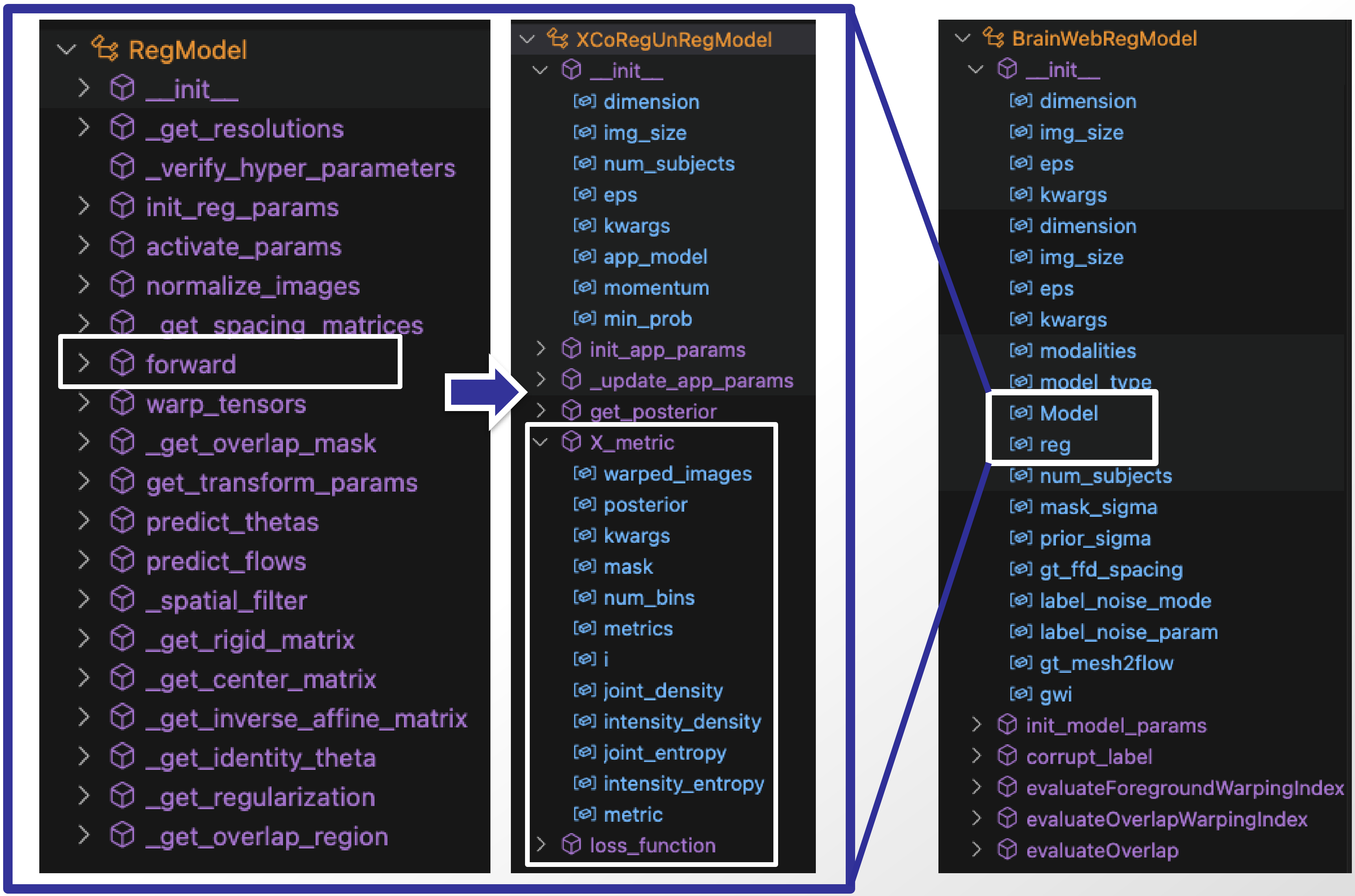
Code: Trainer


```python
for idx, data in enumerate(train_loader):
    pair_names = train_dataset.get_image_name(idx)
    ffd_names = [os.path.basename(name[-1])[:-7] for name in pair_names]
    indices.append('&'.join(ffd_names))

    images = data['images'].to(device)
    ffds = data['ffds'].to(device)
    modalities = data['modalities']
    affines = data['affines']
    headers = data['headers']

    model.init_model_params(images, ffds, phantom)

    opts = [self._get_optimizer([model.reg.params[model.reg.reg_level_type[j]]], lr=lr[j])
            for j in range(model.reg.num_reg_levels)]

    with torch.no_grad():⋯

    for j in range(model.reg.num_reg_levels):
        model.reg.activate_params([j])
        opt = opts[j]
        for step in range(0 if j == 0 else cum_steps[j - 1], cum_steps[j]):

            opt.zero_grad()

            warped_images = model.reg()

            loss = model.reg.loss_function(warped_images)
            loss.backward()

            opt.step()

            if step % display_step == (display_step - 1):⋯
```

- Trainer
  - __init__
  - _get_writer
  - _get_optimizer
  - _get_scheduler
  - train

- IterGroupRegTrainer
  - train
  - _output_minibatch_stats
  - store_predictions
  - reorder_posterior

Code: Model


RegModel
__init__
_get_resolutions
_verify_hyper_parameters
init_reg_params
activate_params
normalize_images
_get_spacing_matrices
forward
warp_tensors
_get_overlap_mask
get_transform_params
predict_thetas
predict_flows
_spatial_filter
_get_rigid_matrix
_get_center_matrix
_get_inverse_affine_matrix
_get_identity_theta
_get_regularization
_get_overlap_region
XCoRegUnRegModel
__init__
dimension
img_size
num_subjects
eps
kwargs
app_model
momentum
min_prob
init_app_params
_update_app_params
get_posterior
X_metric
warped_images
posterior
kwargs
mask
num_bins
metrics
i
joint_density
intensity_density
joint_entropy
intensity_entropy
metric
loss_function
BrainWebRegModel
__init__
dimension
img_size
eps
kwargs
dimension
img_size
eps
kwargs
modalities
model_type
Model
reg
num_subjects
mask_sigma
prior_sigma
gt_ffd_spacing
label_noise_mode
label_noise_param
gt_mesh2flow
gwi
init_model_params
corrupt_label
evaluateForegroundWarpingIndex
evaluateOverlapWarpingIndex
evaluateOverlap

Code: Model

Init parameters

```python
def init_model_params(self, images, ffds, label):
    B = images.shape[0]
    assert images.shape[1] == self.num_subjects

    self.gt = label
    self.label = self.corrupt_label(label, mode=self.label_noise_mode, param=self.label_noise_param)
    self.gt_flows = []
    for i in range(self.num_subjects):
        if self.reg.group2ref and self.reg.inv_warp_ref and i == 0: ...
        else:
            self.gt_flows.append(self.gt_mesh2flow(ffds[:, i]))

    spatial_transformer = SpatialTransformer(size=label.shape[2:], padding_mode='zeros').to(label.device, label.dtype)
    self.inv_warped_labels = [spatial_transformer(self.label, flows=self.gt_flows[i], interp_mode='nearest') ...
    self.inv_warped_gts = [spatial_transformer(label, flows=self.gt_flows[i], interp_mode='nearest') ...
    self.inv_warped_images = [spatial_transformer(images[:, i], flows=self.gt_flows[i])
                              for i in range(self.num_subjects)]

    self.reg.init_reg_params(images=self.inv_warped_images)

    if label is None: ...
    else:
        # label.shape = (B, 4, 217, 181)
        if self.mask_sigma == -1:
            mask = torch.ones(B, 1, *self.reg.img_size, dtype=images.dtype, dev
        else: ...

        if self.prior_sigma == -1:
            prior = torch.full((B, self.reg.num_classes, *self.reg.img_size),
                               fill_value=1 / self.reg.num_classes, device=ima

        else: ...
    self.reg.mask = mask
    self.reg.prior = prior

    if self.model_type == 'XCoRegUn':
        self.reg.init_app_params()
    if self.model_type == 'XCoRegGT':
        self.reg.init_app_params(labels=self.inv_warped_labels)
```

The transformations ϕ are initialized to be the identity, and the prior proportions and the spatial distribution are initialized by

$$\pi_k^{[0]} \triangleq \frac{1}{K}, \qquad \gamma_{\boldsymbol{x},k}^{[0]} \triangleq \frac{\exp(g_{\boldsymbol{x},k})}{\sum_{l=1}^{K} \exp(g_{\boldsymbol{x},l})},$$

where $g_{\boldsymbol{x},k} \sim \mathcal{N}(0,1)$ for $k = 1, \ldots, K$ and $\forall \boldsymbol{x} \in \Omega$.

```python
def init_app_params(self, images=None, template=None, T=0):
    if images is None:
        images = self.forward()

    self.pi = torch.full(size=(self.B, self.num_classes, *[1] * self.dimension),
                         fill_value=1 / self.num_classes, dtype=self.dtype, device=self.device)

    self.template = template
    if self.template is None:
        self.posterior = torch.randn(self.B, self.num_classes, *self.img_size,
                                     dtype=self.dtype, device=self.device).softmax(dim=1)
        self.template = self.posterior.clone()
    else:
        self.posterior = self.template.clone()

    for _ in range(T):
        self._update_app_params(images)

    return
```

Code: Model

大数据学院
School of Data Science

```python
def loss_function(self, warped_images, **kwargs):
    num_bins = kwargs.pop('num_bins', self.num_bins)
    with torch.no_grad():
        overlap_region = self._get_overlap_region()
        mask = torch.logical_and(self._get_overlap_mask(), overlap_region).to(self.dtype)
        mask = F.interpolate(mask, scale_factor=self.scale_factor)

        self._update_app_params(warped_images, mask, num_bins=num_bins)

    post_X_metric = self.X_metric(warped_images, self.template, mask=mask, num_bins=num_bins)
    loss = - post_X_metric

    loss += self._get_regularization()

    return loss
```

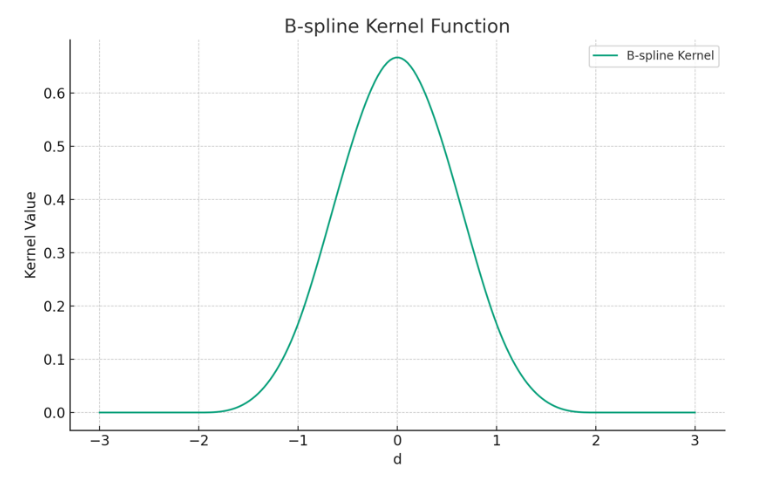
Code: Model

$$f_{jk}^{[t]}\left(\mu_j;\phi_j^{[t]},\mathbf{\Gamma}^{[t]}\right)=P\left(u_{\boldsymbol{x},j}^{\phi_j^{[t]}}=\mu_j \mid z_{\boldsymbol{x},k}=1;\phi_j^{[t]},\mathbf{\Gamma}^{[t]}\right)\triangleq\frac{1}{\mathcal{Z}_{jk}}\sum_{\boldsymbol{x}\in\Omega_S^{\phi_j^{[t]}}}\beta_3\left(\frac{u_{\boldsymbol{x},j}^{\phi_j^{[t]}}-\mu_j}{h}\right)\cdot\gamma_{\boldsymbol{x},k}^{[t]}$$

```python
warped_app_maps = []
for i in range(self.num_subjects):
    warped_app_maps.append(self.app_model(warped_images[i], weight=self.template.detach(), mask=mask,
                                          num_bins=num_bins))
self.posterior = self.get_posterior(torch.stack(warped_app_maps, dim=1))
self.pi = self.posterior.mul(mask).sum(
    dim=tuple(range(2, 2 + self.dimension)), keepdim=True).div(mask.sum(
    dim=tuple(range(2, 2 + self.dimension)), keepdim=True)
).detach()
```

$$\gamma_{\boldsymbol{x},k}^{[t+1]}=q_{\boldsymbol{x},k}^{[t]}=\frac{\pi_k^{[t]}\prod_{j=1}^{N}f_{jk}^{[t]}\left(\mu_j;\phi_j^{[t]},\mathbf{\Gamma}^{[t]}\right)}{\sum_{l=1}^{K}\pi_l^{[t]}\prod_{j=1}^{N}f_{jl}^{[t]}\left(\mu_j;\phi_j^{[t]},\mathbf{\Gamma}^{[t]}\right)}$$

$$\pi_k^{[t+1]}=\frac{\sum_{\boldsymbol{x}\in\Omega^\phi}\gamma_{\boldsymbol{x},k}^{[t+1]}}{\sum_{k=1}^{K}\sum_{\boldsymbol{x}\in\Omega^\phi}\gamma_{\boldsymbol{x},k}^{[t+1]}}=\frac{\sum_{\boldsymbol{x}\in\Omega^\phi}q_{\boldsymbol{x},k}^{[t]}}{\left|\mathbf{\Omega}^{\phi^{[t]}}\right|}$$



Code: Model

$$p_j^{[t+1]}(\mu_j, k; \phi_j) \triangleq \frac{1}{\mathcal{Z}_j} \sum_{x \in \Omega_{S'}^{\phi}} \beta_3 \left(\frac{u_{\boldsymbol{x},j}^{\phi_j} - \mu_j}{h} \right) \cdot \gamma_{\boldsymbol{x},k}^{[t+1]}$$

```
def X_metric(self, warped_images, posterior, **kwargs):
    mask = kwargs.pop('mask', None)
    num_bins = kwargs.pop('num_bins', self.num_bins)

    metrics = []
    for i in range(1 if self.group2ref else 0, self.num_subjects):
        joint_density, _, _ = self.app_model(warped_images[i], weight=posterior, mask=mask,
                                             return_density=True, num_bins=num_bins)
        # sum at the label dimension k
        intensity_density = joint_density.sum(dim=1)

        joint_entropy = - torch.sum(joint_density * joint_density.clamp(min=self.eps).log(), dim=(1, 2)).mean()
        intensity_entropy = - torch.sum(intensity_density * intensity_density.clamp(min=self.eps).log(),
                                        dim=-1).mean()

        metrics.append(- joint_entropy + intensity_entropy)

    metric = torch.sum(torch.stack(metrics))

    return metric
```

$$p_j^{[t+1]}(\mu_j; \phi_j) = \sum_{k=1}^{K} p_j^{[t+1]}(\mu_j, k; \phi_j)$$

与更新 ϕ_j 无关 实现中省略

$$\mathcal{X}(\boldsymbol{U}, \boldsymbol{Z}) = \sum_{j=1}^{N} I(\boldsymbol{U}_j, \boldsymbol{Z}) = \sum_{j=1}^{N} [H(\boldsymbol{U}_j) + \boxed{H(\boldsymbol{Z})} - H(\boldsymbol{U}_j, \boldsymbol{Z})]$$

Code: Model


```python
def _get_regularization(self):
    r = 0
    for j in self.activated_reg_levels:
        if self.reg_level_type[j] not in ['TRA', 'AFF', 'RIG']:
            r += self.bending_energy(self.params[self.reg_level_type[j]]) * self.group_num

    return r
```

图像配准中的弯曲能量（Bending Energy），用于正则化（regularization）优化过程中的位移场（displacement field），用于使变换场更加平滑和连续，有助于避免过度拟合和减少不必要的局部变形。

- 在二维情况下：

  $E_{bending} = \alpha \cdot \mathrm{mean}(||\frac{\partial^2 u}{\partial x^2}||^2 + ||\frac{\partial^2 u}{\partial y^2}||^2 + 2 \cdot ||\frac{\partial^2 u}{\partial x \partial y}||^2)$

- 在三维情况下：

  $E_{bending} = \alpha \cdot \mathrm{mean}(||\frac{\partial^2 u}{\partial x^2}||^2 + ||\frac{\partial^2 u}{\partial y^2}||^2 + ||\frac{\partial^2 u}{\partial z^2}||^2 + 2 \cdot ||\frac{\partial^2 u}{\partial x \partial y}||^2 + 2 \cdot ||\frac{\partial^2 u}{\partial x \partial z}||^2 + 2 \cdot ||\frac{\partial^2 u}{\partial y \partial z}||^2)$

其中：

- u 是位移场（displacement field）
- α 是正则化参数
- $\frac{\partial^2 u}{\partial x^2}$、$\frac{\partial^2 u}{\partial y^2}$、$\frac{\partial^2 u}{\partial z^2}$ 分别代表位移场 u 关于 x、y、z 方向的二阶偏导数
- $\frac{\partial^2 u}{\partial x \partial y}$、$\frac{\partial^2 u}{\partial x \partial z}$、$\frac{\partial^2 u}{\partial y \partial z}$ 分别代表位移场 u 关于 $x-y$、$x-z$、$y-z$ 方向的混合偏导数

```python
class BendingEnergy(LocalDisplacementEnergy):
    def __init__(self, alpha=1, **kwargs):
        super(BendingEnergy, self).__init__(**kwargs)
        self.alpha = alpha

    def forward(self, flow):
        dfdx = self._gradient_txyz(flow, self._gradient_dx)
        dfdy = self._gradient_txyz(flow, self._gradient_dy)

        dfdxx = self._gradient_txyz(dfdx, self._gradient_dx)
        dfdyy = self._gradient_txyz(dfdy, self._gradient_dy)
        dfdxy = self._gradient_txyz(dfdx, self._gradient_dy)

        if self.dimension == 2:
            return self.alpha * torch.mean(dfdxx ** 2 + dfdyy ** 2 + 2 * dfdxy ** 2)

        elif self.dimension == 3:
            dfdz = self._gradient_txyz(flow, self._gradient_dz)
            dfdzz = self._gradient_txyz(dfdz, self._gradient_dz)
            dfdyz = self._gradient_txyz(dfdy, self._gradient_dz)
            dfdxz = self._gradient_txyz(dfdx, self._gradient_dz)

            return self.alpha * torch.mean(
                dfdxx ** 2 + dfdyy ** 2 + dfdzz ** 2 + 2 * dfdxy ** 2 + 2 * dfdxz **

        else:
            raise NotImplementedError
```

Code: Metric


```python
# groupwise warping index
class GWI(nn.Module):
    def __init__(self, unbiased=True, **kwargs):
        super(GWI, self).__init__()
        self.unbiased = unbiased

        self.transform = SpatialTransformer(**kwargs)

    def forward(self, init_flows, pred_flows, masks=None):
        assert init_flows.shape == pred_flows.shape
        if masks is not None:⋯
        n = init_flows.shape[1]
        b = init_flows.shape[0]

        with torch.no_grad():
            if masks is None:⋯
            else:
                masks = masks.to(init_flows.dtype)
            masks_warped = self.transform(rearrange(masks, 'B M ... -> (B M) ...'),
                                          rearrange(pred_flows, 'B M ... -> (B M) ...'), mode='nearest')
            masks_warped = rearrange(masks_warped, '(B M) ... -> B M ...', M=n)

            res = self.transform.getComposedFlows(
                flows=[rearrange(pred_flows, 'B M ... -> (B M) ...'),
                       rearrange(init_flows, 'B M ... -> (B M) ...')])
            res = rearrange(res, '(B M) ... -> B M ...', M=n)
            if self.unbiased:
                res -= torch.mean(res, dim=1, keepdim=True)
                init_flows -= torch.mean(init_flows, dim=1, keepdim=True)

            dims_sum = list(range(2, init_flows.ndim))
            pre_gWI = torch.sqrt(torch.sum((init_flows**2) * masks, dim=dims_sum) / masks.sum(dim=dims_sum))
            post_gWI = torch.sqrt(torch.sum((res**2) * masks_warped, dim=dims_sum) / masks_warped.sum(dim=di
        assert list(pre_gWI.shape) == list(post_gWI.shape) == [b, n]
```

$$\mathrm{gWI}\left(\phi^{\dagger}, \widehat{\phi}\right) \triangleq \frac{1}{N} \sum_{j=1}^{N} \sqrt{\frac{1}{\left|\widehat{\Omega}_j^f\right|} \sum_{\boldsymbol{x} \in \widehat{\Omega}_j^f} \left\|\bar{\boldsymbol{r}}_j(\boldsymbol{x})\right\|_2^2}$$

- BrainWebRegModel
  - __init__
  - corrupt_label
  - dimension
  - eps
  - evaluateForegroundWarpingIndex
  - evaluateOverlap
  - evaluateOverlapWarpingIndex

The root mean squared residual displacement error (**The groupwise warping index (gWI)**)

$$\widehat{\Omega}_j^f \triangleq \left\{x \in \Omega \mid \phi_j^{\dagger} \circ \widehat{\phi}_j(x) \in F\right\}$$

$$\bar{\boldsymbol{r}}_j(\boldsymbol{x}) \triangleq \boldsymbol{r}_j(\boldsymbol{x}) - \frac{1}{N} \sum_{j'=1}^{N} \boldsymbol{r}_{j'}(\boldsymbol{x}),$$

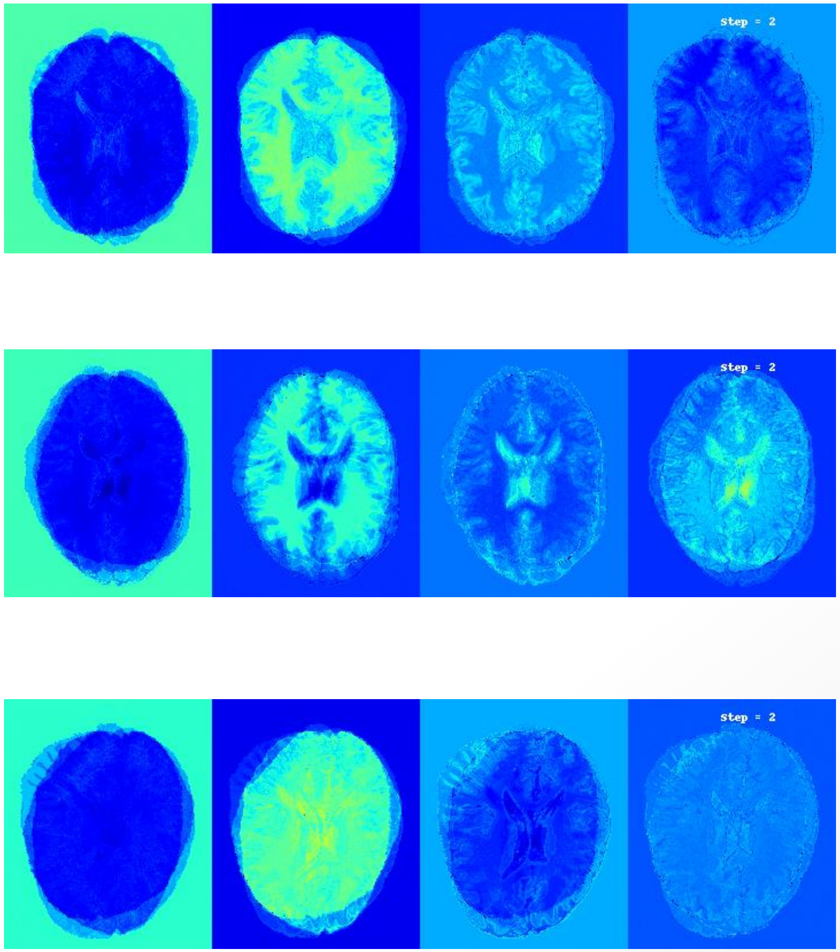
$$\boldsymbol{r}_j(\boldsymbol{x}) \triangleq \phi_j^{\dagger} \circ \widehat{\boldsymbol{\phi}}_j(\boldsymbol{x}) - \boldsymbol{x}.$$

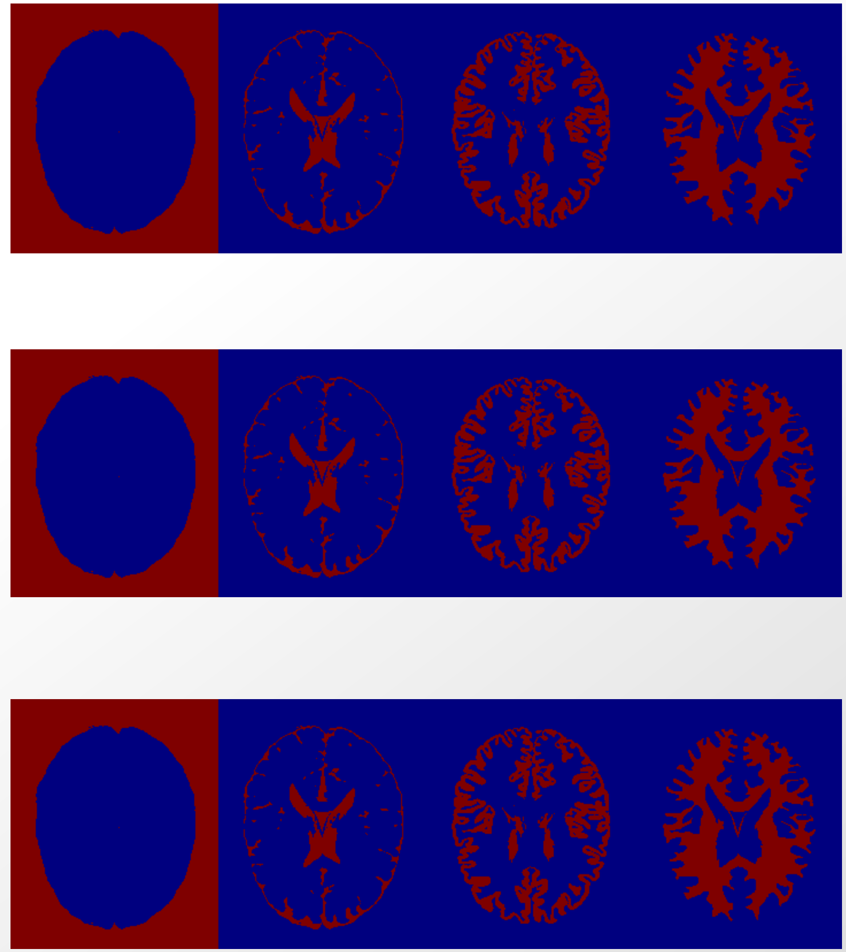
Experiment 1: Group Registration

大数据学院
School of Data Science

Posterior $P\left(z_{\boldsymbol{x},k}=1 \mid \boldsymbol{u}_{\boldsymbol{x}}^{\phi}=\boldsymbol{\mu};\boldsymbol{\theta}^{[t]}\right)$

Label (Not included in training)





CSF GM WM

Initial: 20mm, Reg FFD mesh spacing: 20

Experiment 1: Group Registration

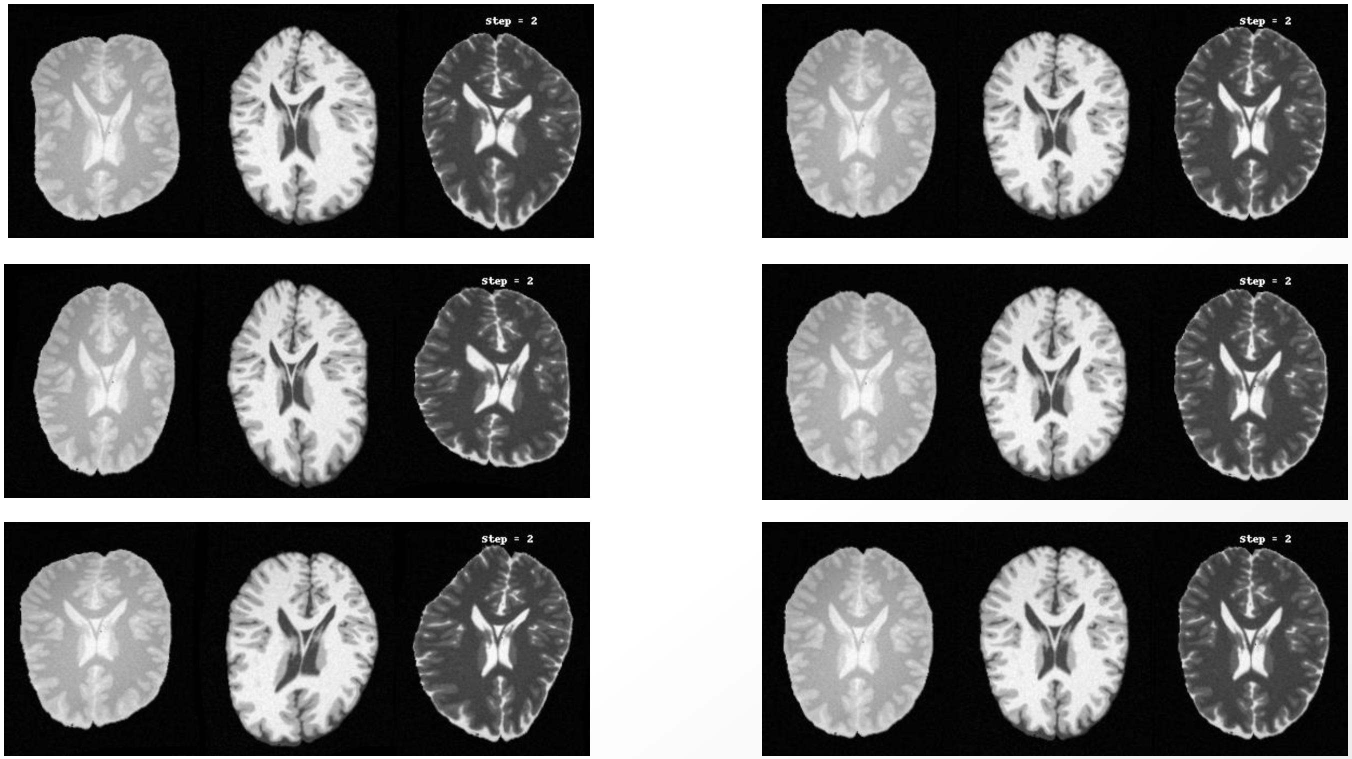


Table 1: Registration parameters and results for experiments on the BrainWeb dataset.

Random FFD level d	Reg FFD mesh spacing	Foreground WI (mm)	PD DSC	T1 DSC	T2 DSC
20mm	20mm	2.292±1.598	76.8±7.7	78.8±8.0	79.8±6.6
15mm	20mm	1.421±0.765	79.4±4.6	80.4±5.4	81.2±4.8
10mm	40mm	0.361±0.102	87.4±1.2	88.0±0.6	88.3±0.7
5mm	40mm	0.252±0.053	88.7±1.1	89.0±0.7	89.2±0.6

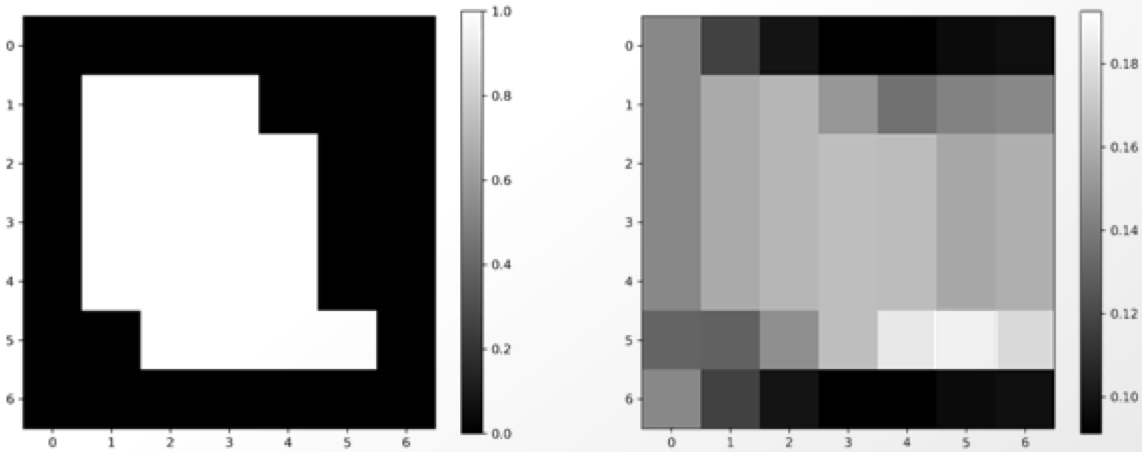
Experiment 2: Deep combined computing

大数据学院
School of Data Science

- Achieving simultaneous registration and segmentation in an end-to-end fashion.
- The extended framework on deep combined computing for multi-sequence cardiac MRI from the MS-CMR dataset.
- **MS-CMR dataset** provides multi-sequence cardiac MR images for 45 patients, **LGE, bSSFP, and T2-weighted**
- **K = 4 for the myocardium, left ventricle, right ventricle and the background**
- Preprocessed
- 39 image slices for training, 15 for validation and 44 for testing
- Five synthetic FFDs were generated with four different mesh spacings for each sequence. 39 * 5^3 * 4 = 19500 image groups

Code: Data


```
def get_distance_prob(one_hot_label, clip_value=50, rho=0.1):
    C = one_hot_label.shape[0]
    dist_map = []
    for c in range(C):
        mask = one_hot_label[c].astype(np.bool_)
        pos_dist = distance(mask)
        neg_dist = distance(~mask)

        dist = pos_dist - neg_dist
        dist_map.append(dist)

    dist = np.stack(dist_map)
    prob = get_normalized_prob(np.exp(np.clip(rho * dist, - clip_value, clip_value)), mode='np', dim=0)

    return prob.astype(np.float32)
```



Signed distance map

$$\tilde{\rho}_{j\phi_j(\boldsymbol{x}),k} \propto \exp[\tau \cdot D_{jk}(\phi_j(\boldsymbol{x}))]$$

Code: Training pipeline

大数据学院
School of Data Science

```python
opts = [self._get_optimizer([{'params': model.net.encoder.parameters()},
                             {'params': model.net.seg_decoder.parameters()},
                             {'params': model.net.seg_output_conv.parameters()}], lr=self.lr[0]),
        self._get_optimizer([{'params': model.net.reg_decoder.parameters()},
                             {'params': model.net.reg_trans.parameters()},
                             {'params': model.net.reg_output_convs.parameters()}], lr=self.lr[1])]
train_metrics = {"Loss": {}}
train_metrics.update(dict([("Post-reg %s Dice" % m.upper(), {}) for m in model.modalities]))

if valid_dataset is not None:…
if test_dataset is not None:…

criterion = self.net.loss_function

max_reg_dice = float('-inf')
for epoch in range(epochs[0]):
    model.train()
    running_loss = 0

    for idx, data in enumerate(train_loader):
        step = epoch * training_iters + idx

        j = (step // inter_steps) % 2    # 交替更新
        opt = opts[j]

        opt.zero_grad(set_to_none=True)

        images = data['images'].to(device)
        ffds = data['ffds']
        if isinstance(ffds, torch.Tensor):
            ffds = ffds.to(device)
        probs = data['probs']
        if isinstance(probs, torch.Tensor):
            probs = probs.to(device)
        atlas_prob = data['atlas_prob']
        if isinstance(atlas_prob, torch.Tensor):
            atlas_prob = atlas_prob.to(device)

        _ = model(images, init_flows=ffds)
        loss = criterion(probs=probs, atlas_prob=atlas_prob)

        loss.backward()
```

inter_step = 5

Code: Network


```python
def forward(self, images, init_flows=None):
    self.init_flows = init_flows
    if init_flows is not None:
        self.images = self.transform_images(images, init_flows)[1]
    x = self._normalize_images(torch.stack(self.images, dim=1))

    flows, self.seg_probs = self.net(x)

    self.flows = flows - torch.mean(flows, dim=1, keepdim=True)

    self.warped_images = self.transform_images(self.images, self.flows)[1]

    return self.warped_images
```

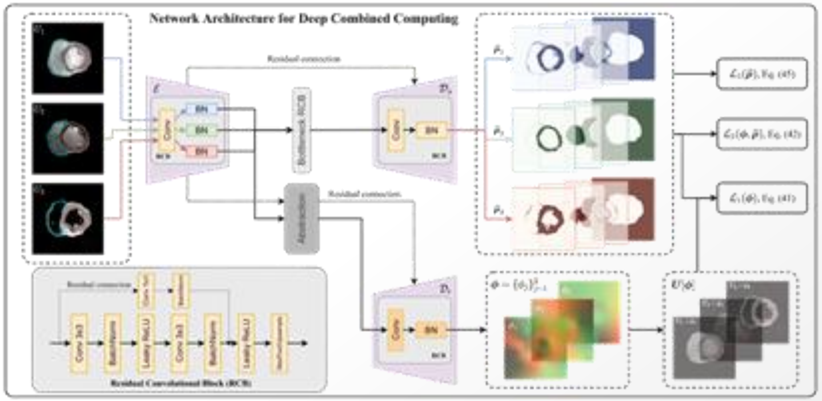


Fig. 5. An example of the network architecture for deep combined computing when $N = 3$. The encoder $\mathcal{E}$, the decoders $\mathcal{D}_s$ and $\mathcal{D}_r$, and the bottleneck are composed of residual convolutional blocks. Domain-specific batch normalization layers are indicated with different colours. Cardiac structures, *i.e.* myocardium, left ventricle (LV) and right ventricle (RV), are rendered as contours.

```python
def forward(self, x):
    x, x_enc = self.encoder(x)

    seg_dec = self.seg_decoder(x=rearrange(x, 'B M ... -> (B M) ...'),
                               x_enc=[rearrange(y, 'B M ... -> (B M) ...') for y in x_enc])
    seg = rearrange(self.seg_output_conv(seg_dec[0]), '(B M) ... -> B M ...', M=len(self.modalities))

    x_trans = self.reg_trans['trans_%s' % self.num_blocks](self.fuse(x))
    x_enc_trans = [self.reg_trans['trans_%s' % i](self.fuse(x_enc[i])) for i in range(self.num_blocks)]

    x_dec = self.reg_decoder(x_trans, x_enc_trans)

    flows = []
    for i in range(self.num_blocks):
        flows.append(self.resize(self.reg_output_convs['output_conv_%s' % i](x_dec[i]), factor=2 ** i))

    if self.compose_ddf:
        y = (SpatialTransformer(size=x.shape[3:], padding_mode='zeros').getComposedFlows(flows[::-1]))
    else:
        y = (torch.stack(flows).sum(dim=0))
    y = rearrange(y, 'B (M d) ... -> B M d ...', d=self.dimension)

    return y, torch.softmax(seg, dim=2)
```

- Encoder
- Seg_decoder
- Reg_decoder

Code: Loss function

大数据学院
School of Data Science

```python
def loss_function(self, probs=None, **kwargs):
    atlas_prob = kwargs.pop('atlas_prob', None)
    if self.use_atlas: ...
    else: ...

    self.mask = self.get_overlap_region()

    warped_probs = self.transform_probs(probs, detach_seg_flows=False)

    with torch.no_grad():
        init_posterior = self.get_posterior(self.warped_images, warped_probs=warped_probs,
                                            warped_atlas=warped_atlas, use_probs=True)
        self.posterior = self.update_posterior(self.warped_images, init_posterior,
                                               warped_probs=warped_probs, warped_atlas=warped_atlas,
                                               use_probs=True, T=self.update_steps)

    loss = self.X_metric(self.images, probs=self.seg_probs)

    loss += self.X_metric(self.warped_images, posterior=self.posterior)

    loss += self.seg_loss_posterior(warped_probs, self.posterior, warped_atlas=warped_atlas)

    loss += self.seg_loss_probs(self.seg_probs, probs)

    loss += self.bending_energy(self.flows) * self.num_images

    return loss
```

Code: Loss function


```python
def get_posterior(self, warped_images=None, warped_probs=None, warped_atlas=None, prior=None, posterior=None,
                  use_probs=False):
    if isinstance(warped_images, torch.Tensor):
        warped_images = torch.unbind(warped_images, dim=1)
    if isinstance(warped_probs, torch.Tensor):
        warped_probs = torch.unbind(warped_probs, dim=1)

    if warped_images is not None: ⋯
    elif warped_probs is not None: ⋯
    else: ⋯

    if hasattr(self, 'mask'):
        mask = self.mask
    else:
        mask = self.get_overlap_region()

    if prior is None: ⋯

    warped_cpds = []
    if warped_probs is None: ⋯
    else:
        if use_probs:
            for i in range(self.num_subjects):
                if i in self.sup_idx:
                    prob = warped_probs[i]
                    if self.clamp_prob:
                        prob = utils.get_normalized_prob(torch.clamp(prob, self.prob_interval[0],
                                                                     self.prob_interval[1]), dim=1)
                    warped_cpds.append(prob)

    if self.use_atlas: ⋯

    if warped_images is not None:
        for i in range(self.num_subjects):
            warped_cpds.append(self.app_model(warped_images[i], weight=warped_probs[i], mask=mask))

    posterior = utils.get_normalized_prob(torch.clamp_min(torch.stack(warped_cpds, dim=1),
                                                          self.eps).log().sum(dim=1).exp().mul(prior),
                                          dim=1)

    return posterior
```

计算后验

$$q_{\boldsymbol{x},k}^{[t]} \propto \pi_k^{[t]} \prod_{j=1}^{N} f_{jk}^{[t]}(\mu_j)$$

Appearance model

$$f_{jk}^{[0]}(\mu_j) \propto \sum_{\omega\in\Omega_j} \beta_3\left(\frac{U_j(\boldsymbol{\omega}) - \mu_j}{h}\right) \cdot \rho_{jk}(\boldsymbol{\omega})$$

$$f_{jk}^{[1]}(\mu_j) \propto \sum_{\boldsymbol{x}\in\Omega_S^{\phi_j^{[t]}}} \beta_3\left(\frac{u_{\boldsymbol{x},j}^{\phi_j^{[1]}} - \mu_j}{h}\right) \cdot \gamma_{\boldsymbol{x},k}^{[1]}$$

Code: Loss function

$$\mathcal{L}_1(\phi) \triangleq -\mathcal{X}\left(\boldsymbol{U}[\phi], \boldsymbol{Z}^{[2]}\right) + \lambda \cdot R(\phi)$$

```
loss += self.X_metric(self.warped_images, posterior=self.posterior)
```

$t = 1$

$$\mathcal{X}(\boldsymbol{U}, \boldsymbol{Z}) = \sum_{j=1}^{N} I(U_j, \boldsymbol{Z}) = \sum_{j=1}^{N} [H(U_j) + H(\boldsymbol{Z}) - H(U_j, \boldsymbol{Z})]$$

$$\mathcal{X}(\boldsymbol{U}, \boldsymbol{Y}) = \sum_{j=1}^{N} I(U_j, Y_j) = \sum_{j=1}^{N} [H(U_j) + H(\boldsymbol{Y_j}) - H(U_j, \boldsymbol{Y_j})]$$

```
def X_metric(self, images, probs=None, posterior=None):
    if isinstance(images, torch.Tensor):
        images = torch.unbind(images, dim=1)
    if isinstance(probs, torch.Tensor):
        probs = torch.unbind(probs, dim=1)
    if probs is None:
        probs = [posterior] * self.num_subjects

    losses = []
    for i in range(self.num_subjects):
        joint_density, _, _ = self.app_model(images[i], weight=probs[i], mask=self.mask, return_density=True)
        intensity_density = joint_density.sum(dim=1)
        class_density = joint_density.sum(dim=2)

        joint_entropy = - torch.sum(joint_density * joint_density.clamp(min=self.eps).log(), dim=(1, 2)).mean()
        intensity_entropy = - torch.sum(intensity_density * intensity_density.clamp(min=self.eps).log(),
                                        dim=-1).mean()
        class_entropy = - torch.sum(class_density * class_density.clamp(min=self.eps).log(), dim=-1).mean()

        losses.append(joint_entropy - intensity_entropy - class_entropy)

    loss = torch.sum(torch.stack(losses))

    return loss
```

Note: We use $Z^{[2]}$ instead of $Z^{[1]}$ because at $t = 0$ the appearance model is calculated using the probability maps of the image anatomy rather than the spatial distribution of the common anatomy.

Code: Loss function

大数据学院 School of Data Science

```
def seg_loss_posterior(self, warped_probs, posterior, warped_atlas=None):
    if isinstance(warped_probs, torch.Tensor):
        warped_probs = torch.unbind(warped_probs, dim=1)

    losses = []
    for i in range(self.num_subjects):
        prob = warped_probs[i]
        loss = self.ce(posterior, prob, mask=self.mask)
        losses.append(loss)

    if warped_atlas is not None:
        losses.append(self.ce(posterior, warped_atlas, mask=self.mask))

    return torch.sum(torch.stack(losses))
```

$$\mathcal{L}_2(\boldsymbol{\phi}, \widehat{\boldsymbol{\rho}}) \triangleq \sum_{j \in \mathcal{J}} H_{\boldsymbol{Z}^{[2]}}(\boldsymbol{Y}_j \circ \phi_j) + \sum_{j \notin \mathcal{J}} H_{\boldsymbol{Z}^{[2]}}\left(\widehat{\boldsymbol{Y}}_j \circ \phi_j\right)$$

Code: Loss function

大数据学院 School of Data Science

```python
def seg_loss_probs(self, seg_probs, probs=None):
    losses = []
    for i in range(self.num_subjects):
        loss = 0.
        if i in self.sup_idx:
            if probs is not None:
                init_prob = self.transform(probs[:, i], flows=self.init_flows[:, i])
                loss += self.ce(init_prob, seg_probs[:, i])
                loss += self.dice_loss(init_prob, seg_probs[:, i])
        else:
            prob = seg_probs[:, i]
            loss += torch.sum(prob * prob.clamp(min=self.eps).log(), dim=1).mean().neg()
        losses.append(loss)

    return torch.sum(torch.stack(losses))
```

```python
loss = self.X_metric(self.images, probs=self.seg_probs)
```

$$\mathcal{L}_3(\widehat{\boldsymbol{\rho}}) \triangleq -\sum_{j=1}^{N} I\left(U_j, \widehat{\boldsymbol{Y}}_j\right) + \sum_{j \notin \mathcal{J}} H\left(\widehat{\boldsymbol{Y}}_j\right) + \sum_{j \in \mathcal{J}} \mathcal{L}_{\text{seg}}\left(\boldsymbol{Y}_j, \widehat{\boldsymbol{Y}}_j\right)$$

$$\mathcal{L}_{\text{seg}}\left(\boldsymbol{Y}_j, \widehat{\boldsymbol{Y}}_j\right) \triangleq H_{\boldsymbol{Y}_j}\left(\widehat{\boldsymbol{Y}}_j\right) + \left[1 - \text{DSC}\left(\boldsymbol{Y}_j, \widehat{\boldsymbol{Y}}_j\right)\right]$$

Experiment 2: Deep combined computing

大数据学院
School of Data Science

TABLE 5
Results on the MS-CMR Dataset

Strategy	Reg DSC	Seg DSC		
		LGE	bSSFP	T2
None	72.2 ± 10.1	—	—	—
DGR	86.2 ± 4.1	—	—	—
MvMM [13]	81.8 ± 8.7	84.5 ± 9.0	84.4 ± 8.5	78.3 ± 14.8
DCC+AT	88.5 ± 3.4	82.0 ± 3.8	81.2 ± 4.5	83.4 ± 4.1
DCC+BS	87.6 ± 4.0	85.8 ± 3.9	89.9 ± 2.8	86.5 ± 4.2
DCC+All	89.5 ± 3.5	92.6 ± 2.0	92.4 ± 3.1	92.7 ± 3.4

The table presents the mean and standard deviation of the registration and segmentation DSCs (%) for deep combined computing using different training strategies and another competing method MvMM.

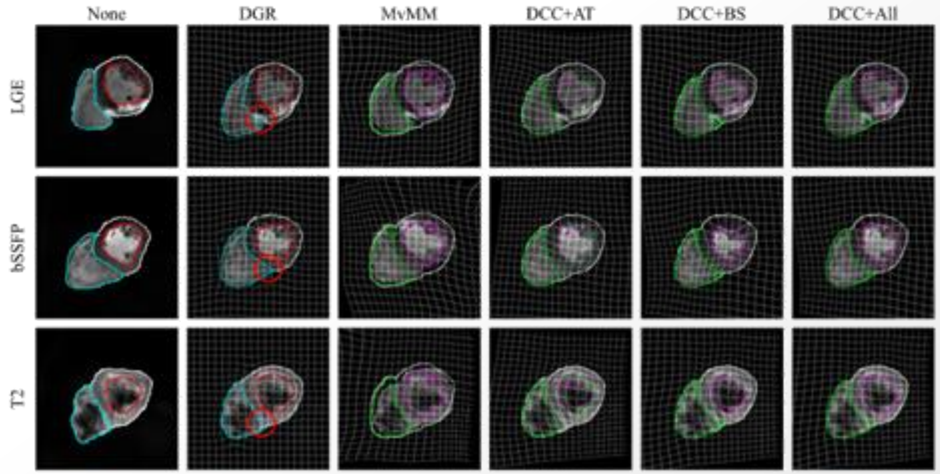


Fig. 10. Results of an exemplar case from the MS-CMR dataset with median Reg DSC before co-registration. Ground-truth segmentation masks are rendered as contours for None and DGR, while posterior segmentation is displayed for MvMM, DCC+AT, DCC+BS and DCC+All. Each column visualizes the registered images from a certain method. Regions with ambiguous intensity class correspondence are indicated by red circles. Readers are referred to the supplementary material, available online or the online version of this paper for details.

- Both registration and segmentation accuracies improve with increased supervision
- Compared to MvMM, the DCC+AT strategy performs better in registration

Acknowledge

Huge thanks to:

Xinzhe Luo

for the help in theoretical content and code implementation!!

Thank You！