



Medical Image Analysis: Multivariate Mixture Model for Myocardial Segmentation Combining Multi-Source Images

Xinzhe Luo

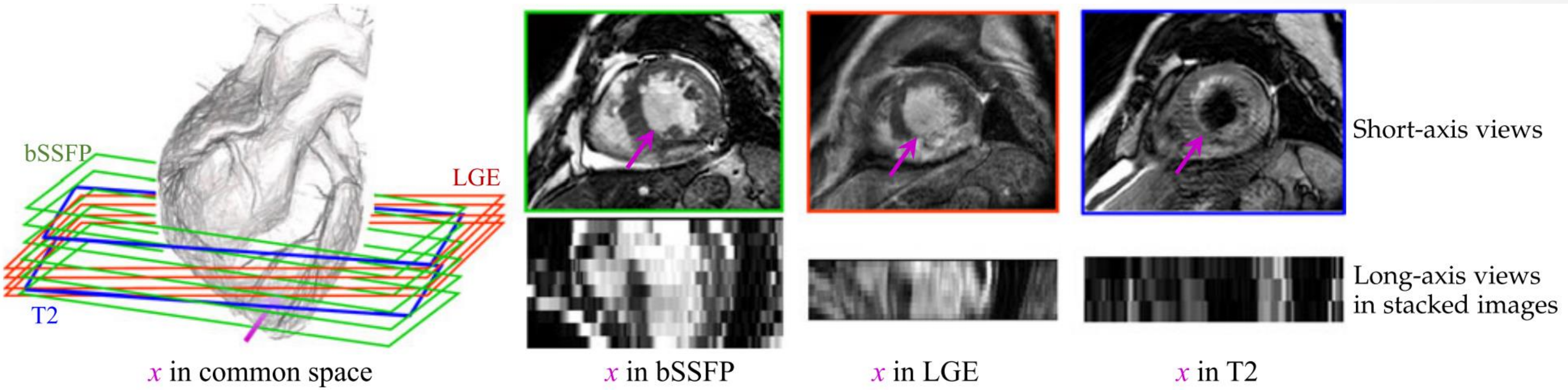
School of Data Science, Fudan University

大数据学院
School of Data Science

Contents

Introduction

- Multi-modality medical images



Zhuang, Xiahai. "Multivariate mixture model for myocardial segmentation combining multi-source images." IEEE transactions on pattern analysis and machine intelligence 41, no. 12 (2019): 2933-2946.

Motivation

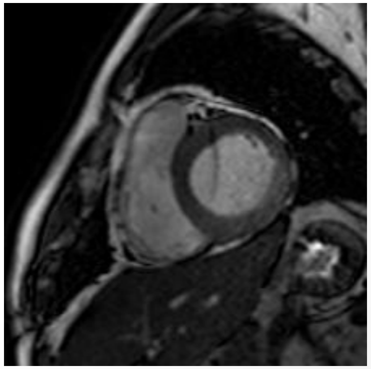
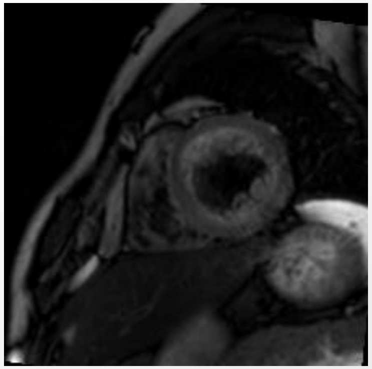
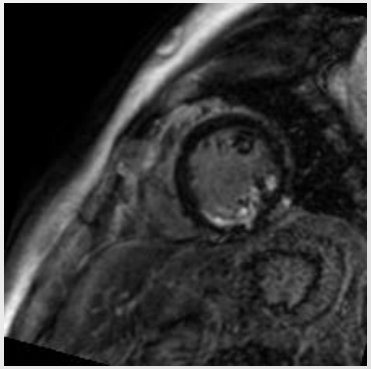
- Myocardial infarction identification from LGE CMR
- Automatic myocardial segmentation from LGE CMR
- Combined segmentation for multi-source images
- Simultaneous registration and segmentation

Contents

大数据学院
School of Data Science

Multivariate mixture model

- **Notations:**
- Denote $\boldsymbol{I} = \{I_i | i = 1, \ldots, N_I\}$ be a set of N_I acquired from the same subject.
- Denote Ω as the common space, which is defined by the combination of images.
- For a location $x \in \Omega$, denote tissue types by labels, $s(x) = k$, $k \in K$, and the subtypes of a tissue k in image I_i as $z_i(x) = c$, $c \in C_{ik}$.

Multivariate mixture model

- **Likelihood function:** $L(\theta; \boldsymbol{I}) = p(\boldsymbol{I}|\theta)$
- Assuming all images are located in the common space
- Assuming independence of each location, one gets

$$p(\boldsymbol{I}|\theta) = \prod_{x \in \Omega} p(\boldsymbol{I}(x)|\theta)$$

- Assuming mixture model on label assignments,

$$p(\boldsymbol{I}(x)|\theta) = \sum_{k \in K} \pi_{kx} \cdot p(\boldsymbol{I}(x)|s(x) = k, \theta)$$

where $\pi_{kx} = p(s(x) = k|\theta)$ is the label proportion.

Multivariate mixture model

- **Likelihood function:**

$$L(\theta; \boldsymbol{I}) = \prod_{x\in\Omega} \sum_{k\in K} \pi_{kx} \cdot p(\boldsymbol{I}(x)|s(x) = k, \theta)$$

- Assuming conditional independence of intensities between different images:

$$p(\boldsymbol{I}(x)|s(x) = k, \theta) = \prod_{i=1}^{N_I} p(I_i(x)|k_x, \theta)$$

- Assuming Gaussian mixture model as the intensity distribution:

$$p(I_i(x)|k_x, \theta) = \sum_{c\in C_{ik}} \tau_{ikc} \cdot \Phi\left(I_i(x); \mu_{ikc}, \sigma_{ikc}^2\right)$$

Multivariate mixture model

- **Likelihood function:**

$$L(\theta;\boldsymbol{I})=\prod_{x\in\Omega}\sum_{k\in K}\pi_{kx}\prod_{i=1}^{N_I}\sum_{c\in C_{ik}}\tau_{ikc}\cdot\Phi\left(I_i(x);\mu_{ikc},\sigma_{ikc}^2\right)$$

where $\theta=\left\{\pi_{kx},\tau_{ikc},\mu_{ikc},\sigma_{ikc}^2\right\}$ are model parameters.

Multivariate mixture model

- **Graphical model:**

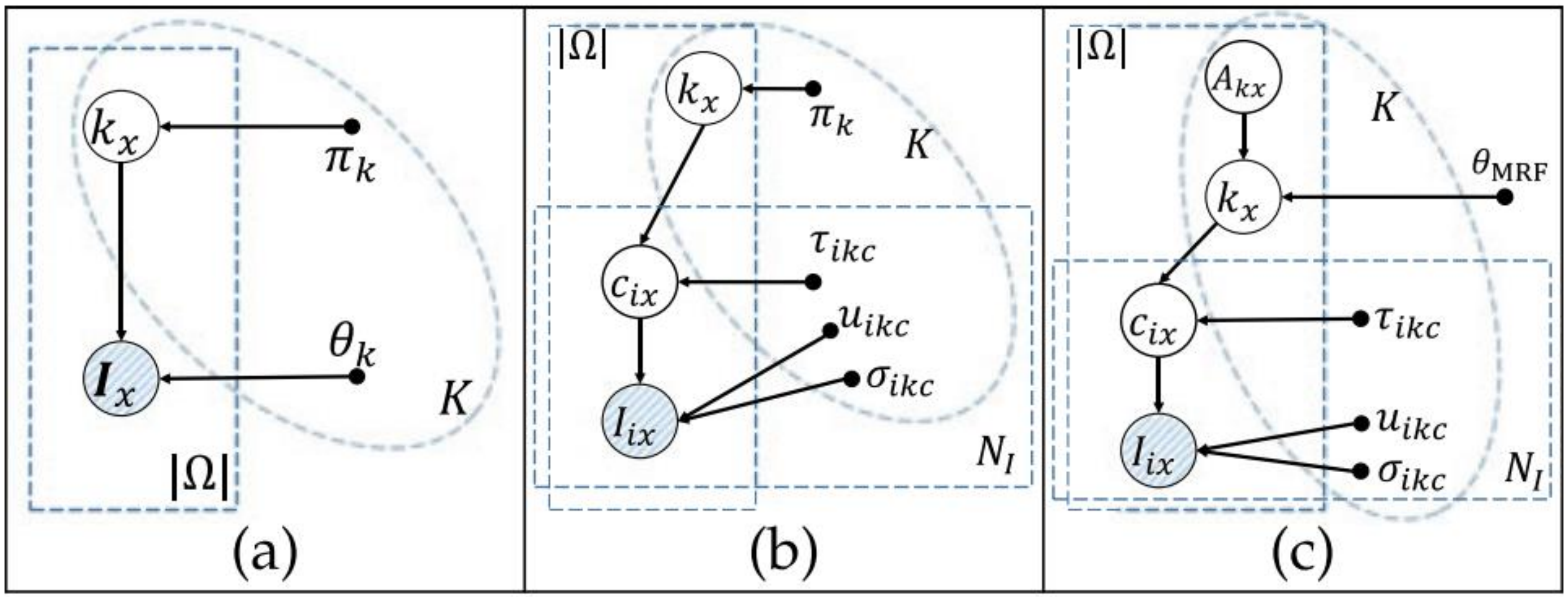
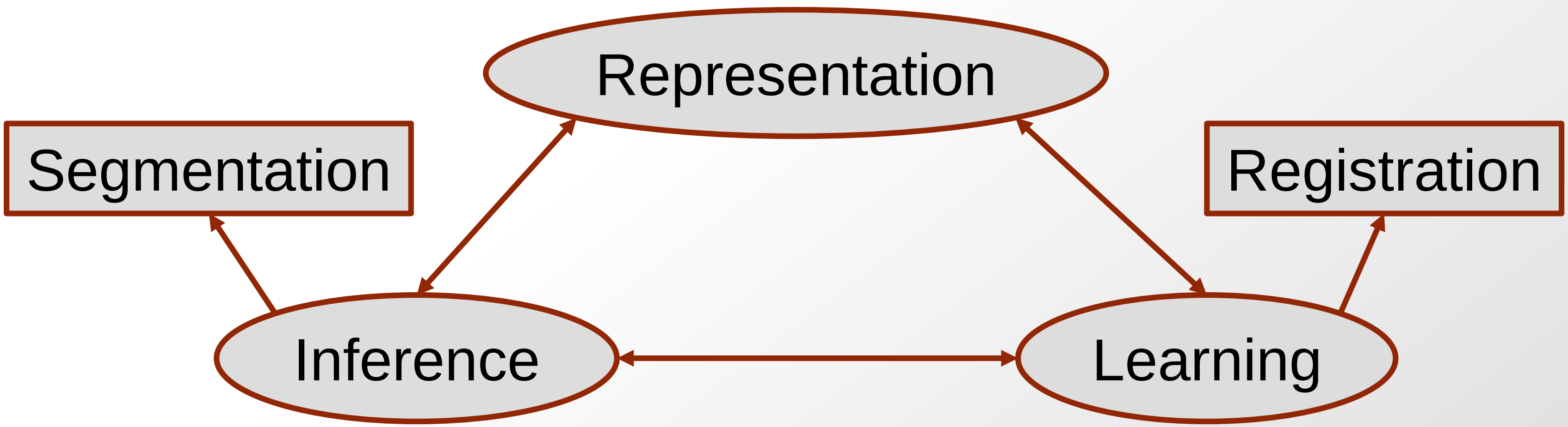


Fig. 3. The graphical representation of the multivariate mixture model in three formulations. Readers are referred to the text for details.

Expectation-maximization algorithm

大数据学院 School of Data Science

- **Objective:**
- Learning parameters of the model from the observed data
  - Data are considered as realizations of the generative model
  - Registration
- Inference given the learned parameters
  - Segmentation



Expectation-maximization algorithm

大数据学院 School of Data Science

- **Maximum likelihood estimator (MLE)**
- Log-likelihood:

$$\ell(\theta; \boldsymbol{I}) = \sum_{x \in \Omega} \ln \left\{ \sum_{k \in K} \pi_{kx} \prod_{i=1}^{N_I} \sum_{c \in C_{ik}} \tau_{ikc} \cdot \Phi\left(I_i(x); \mu_{ikc}, \sigma_{ikc}^2\right) \right\}$$

- **Problems related to MLE:**
  - Maximizing the log-likelihood is not a well posed problem because singularities will occur whenever one of the Gaussian components ‘collapses’ onto a specific data points when we have at least two components in the mixture
  - The presence of the summation over k that appears inside the logarithm make maximization difficult

Bishop, Christopher M. Pattern recognition and machine learning. springer, 2006, Section 9.2.1

Expectation-maximization algorithm

大数据学院 School of Data Science

- **E-step:** compute posterior distribution of the latent variables $\{c_{ik}, k_x\}$ given the current estimate of the model parameters $\theta^{[m]}$
- Posterior of k_x:

$$P_{kx}^{[m+1]} \coloneqq p\left(s(x) = k \middle| \boldsymbol{I}; \theta^{[m]}\right)$$

$$= \frac{p\left(\boldsymbol{I}(x) \middle| k_x; \theta^{[m]}\right)\pi_{kx}^{[m]}}{\sum_{l \in K} p\left(\boldsymbol{I}(x) \middle| l_x; \theta^{[m]}\right)\pi_{lx}^{[m]}}$$

- Posterior of $\{c_{ik}, k_x\}$:

$$P_{ikcx}^{[m+1]} \coloneqq p\left(s(x) = k, z_i(x) = c_{ik} \middle| \boldsymbol{I}, \theta^{[m]}\right)$$

$$= p\left(c_{ikx} \middle| k_x, \boldsymbol{I}, \theta^{[m]}\right) P_{kx}^{[m+1]}$$

$$= \frac{\Phi\left(I_i(x); \mu_{ikc}^{[m]}, \left(\sigma_{ikc}^2\right)^{[m]}\right)\tau_{ikc}^{[m]}}{p\left(I_i(x) \middle| k_x; \theta^{[m]}\right)} P_{kx}^{[m+1]}$$

Expectation-maximization algorithm

- **M-step:** Maximize the expected complete-log-likelihood under the updated posterior distribution with respect to the model parameters
- Log-likelihood (marginal):

$$\ln p(\boldsymbol{I}|\theta) = \sum_{x\in\Omega} \ln \left\{ \sum_{k\in K} \pi_{kx} \prod_{i=1}^{N_I} \sum_{c\in C_{ik}} \tau_{ikc} \cdot \Phi\left(I_i(x); \mu_{ikc}, \sigma_{ikc}^2\right) \right\}$$

- Expected complete-log-likelihood (expected joint):

$\mathbb{E}[\ln p(\boldsymbol{I}, \boldsymbol{Z}, \boldsymbol{S}|\theta)]$

$$= \sum_{x\in\Omega} \left\{ \sum_{k\in K} P_{kx}^{[m+1]} \ln \pi_{kx} + \sum_{k\in K} \sum_{c\in C_{ik}} P_{ikcx}^{[m+1]} \left(\ln \tau_{ikc} + \ln \Phi\left(I_i(x); \mu_{ikc}, \sigma_{ikc}^2\right)\right) \right\}$$

Expectation-maximization algorithm

大数据学院 School of Data Science

- Expected complete-log-likelihood (expected joint):

$$\mathbb{E}[\ln p(\boldsymbol{I},\boldsymbol{Z},\boldsymbol{S}|\theta)]$$

$$= \sum_{x\in\Omega}\left\{\sum_{k\in K} P_{kx}^{[m+1]} \ln \pi_{kx} + \sum_{k\in K}\sum_{c\in C_{ik}} P_{ikcx}^{[m+1]}\left(\ln \tau_{ikc} + \ln \Phi\left(I_i(x);\mu_{ikc},\sigma_{ikc}^2\right)\right)\right\}$$

- **M-step:** Maximizing $\mathbb{E}[\ln p(\boldsymbol{I},\boldsymbol{Z},\boldsymbol{S}|\theta)]$ with respect to $\theta = \{\pi_{kx},\tau_{ikc},\mu_{ikc},\sigma_{ikc}^2\}$ gives

$\pi_{kx}^{[m+1]} = P_{kx}^{[m+1]}$ and $\pi_k^{[m+1]} = \frac{\sum_{x\in\Omega} P_{kx}^{[m+1]}}{\sum_{x\in\Omega}\sum_{k\in K} P_{kx}^{[m+1]}}$ without spatial regularization

$$\tau_{ikc}^{[m+1]} = \frac{\sum_{x\in\Omega} P_{ikcx}^{[m+1]}}{\sum_{x\in\Omega}\sum_{c\in C_{ik}} P_{ikcx}^{[m+1]}}$$

$$\mu_{ikc}^{[m+1]} = \frac{\sum_{x\in\Omega} I_i(x) P_{ikcx}^{[m+1]}}{\sum_{x\in\Omega} P_{ikcx}^{[m+1]}}$$

$$\left(\sigma_{ikc}^2\right)^{[m+1]} = \frac{\sum_{x\in\Omega}\left(I_i(x)-\mu_{ikc}^{[m+1]}\right)^2 P_{ikcx}^{[m+1]}}{\sum_{x\in\Omega} P_{ikcx}^{[m+1]}}$$

Spatial regularization

- **Motivation:** Voxels with the same intensity distribution in medical images can come from different structures.

- **Probabilistic atlases:**

$$\pi_{kx} \propto \pi_k \cdot p(A_{kx})$$

where $p(A_{kx}) = p_A(s(x) = k)$.

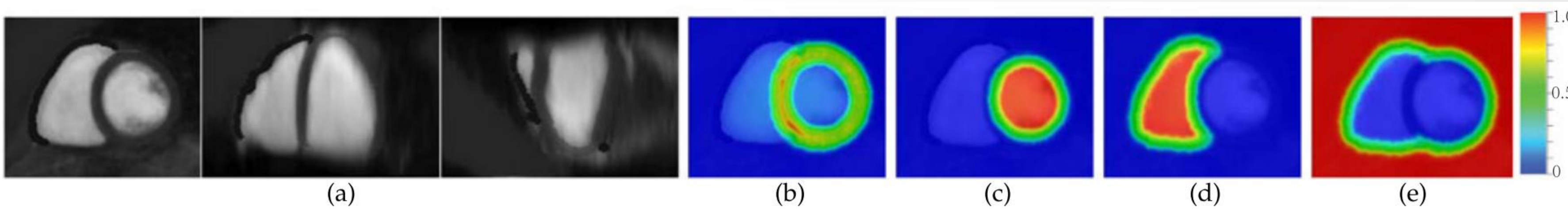


Fig. 4. The three orthogonal views of the atlas intensity image (a), and the short-axis views of the four probabilistic atlases of myocardium (b), left ventricle blood pool (c), right ventricular blood pool (d) and background (e). The probability maps are superimposed onto the intensity image, and the color bars indicate the mapping between the probability values and displayed colors.

- **M-step with probabilistic atlas:** Maximize $\mathbb{E}[\ln p(\boldsymbol{I}, \boldsymbol{Z}, \boldsymbol{S}|\theta)]$ with respect to the proportion parameters $\{\pi_k\}$:

$$\mathbb{E}[\ln p(\boldsymbol{I}, \boldsymbol{Z}, \boldsymbol{S}|\theta)] = \sum_{x\in\Omega}\sum_{k\in K} P_{kx}^{[m+1]} \ln \pi_{kx} + \text{const.}$$

where

$$\sum_{x\in\Omega}\sum_{k\in K} P_{kx}^{[m+1]} \ln \pi_{kx} = \sum_{x\in\Omega}\sum_{k\in K} P_{kx}^{[m+1]} \left[\ln \pi_k p(A_{kx}) - \ln \sum_{j\in K} \pi_j p(A_{jx})\right]$$

Initialization

大数据学院 School of Data Science

- Parameters are initialized based on the atlas prior probabilities

$$\pi_k^{[0]} = \frac{\sum_x p(A_{kx})}{\sum_{l\in K}\sum_x p(A_{lx})}$$

$$\tau_{ikc}^{[0]} = \frac{1}{|C_{ik}|}$$

$$\mu_{ikc}^{[0]} = \begin{cases} \mu_{ik}^{[0]} + a\,\sigma_{ik}^{[0]}, & |C_{ik} \geq 2| \\ \mu_{ik}^{[0]}, & |C_{ik} = 1| \end{cases}$$

$$\left(\sigma_{ikc}^{[0]}\right)^2 = |C_{ik}|\left(\sigma_{ik}^{[0]}\right)^2$$

where $\mu_{ik}^{[0]} = \frac{\sum_x I_i(x)p(A_{kx})}{\sum_x p(A_{kx})}$, and $\left(\sigma_{ik}^{[0]}\right)^2 = \frac{\sum_x\left(I_i(x)-\mu_{ik}^{[0]}\right)^2 p(A_{kx})}{\sum_x p(A_{kx})}$

Generalized EM (GEM)

- The M-step yields the same solution for the segmentation parameters $\theta^{[m+1]}$ as in the case of the independent model

- However, the M-step remains intractable for the MRF parameters Φ. Instead of aiming to maximize $\mathcal{L}(q, \Theta)$ with respect to Φ, the GEM seeks to change the parameters in such a way as to increase its value.

- Expectation conditional maximization (ECM) partitions the parameters into groups, and the M-step is broken down into multiple steps each of which involves optimizing one of the subset with the remainder held fixed.

Registration in MvMM

- **The motion shift of a slice against the common space.**

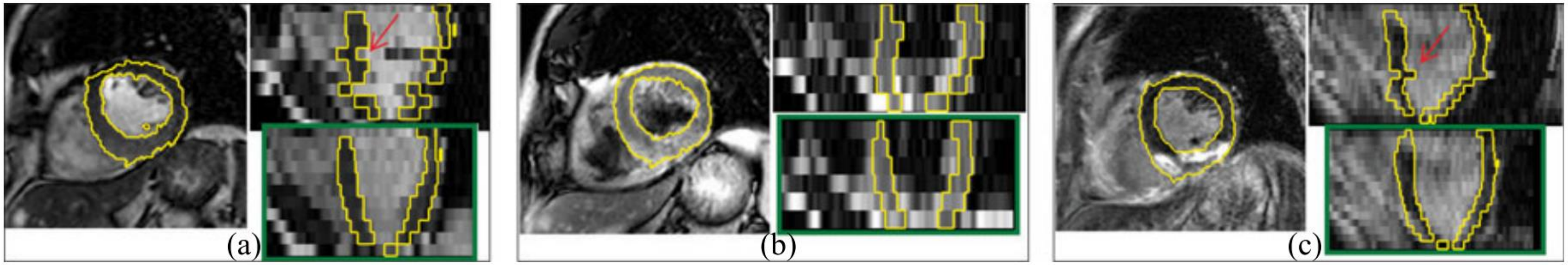


Fig. 5. Segmentation results of the three CMR sequences, (a) bSSFP, (b) T2-weight, (c) LGE; myocardial boundaries are highlighted in yellow color; the motion shifts are pointed out by the red arrows. The images in the green boxes of (a)-(c) are the shift corrected images.

- The motion shift of a slice is modelled by a rigid transformation:

$$p\left(I_i(x)\middle|c_{ik};\theta,G_{i,s}\right)=\Phi_{ikc}\left(I_i\left(G_{i,s}(x)\right)\right)$$

where $\{G_{i,s}\}$ are the transformations for correcting slices.

Registration in MvMM

- **The misalignment of the atlas against the common space**
- Introduce the atlas deformation D for correcting the mis-registration:

$$p_A(s(x)=k|D)=p_A\big(s\big(D(x)\big)=k\big)=A_k\big(D(x)\big),\qquad k=1,\dots,K$$

- **For the independent model, the log-likelihood is reformulated as:**

$$\ell\big(\theta,D,\{G_{i,s}\}\big)=\sum_{x\in\Omega}\ln\left\{\sum_{k\in K}\pi_{kx|D}\prod_{i=1}^{N_I}\sum_{c\in C_{ik}}\tau_{ikc}\cdot\Phi_{ikc}\Big(I_i\Big(G_{i,s}(x)\Big)\Big)\right\}$$

Iterative Conditional Mode (ICM)

大数据学院
School of Data Science

- ICM optimizes the segmentation and registration parameters alternately
  - Update segmentation parameters θ by EM algorithm
    - E−step
    - M−step
  - Update registration parameters $D, \{G_{i,s}\}$ by gradient ascent of the log−likelihood

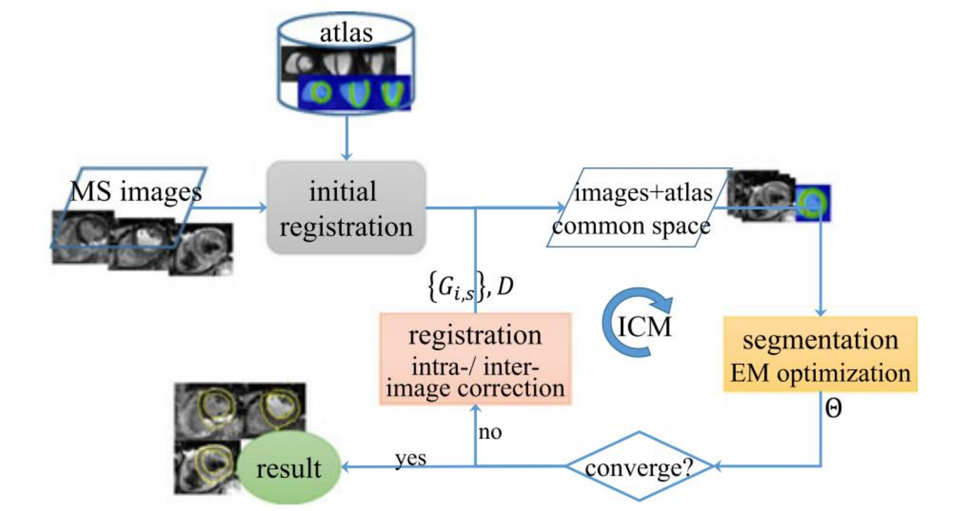
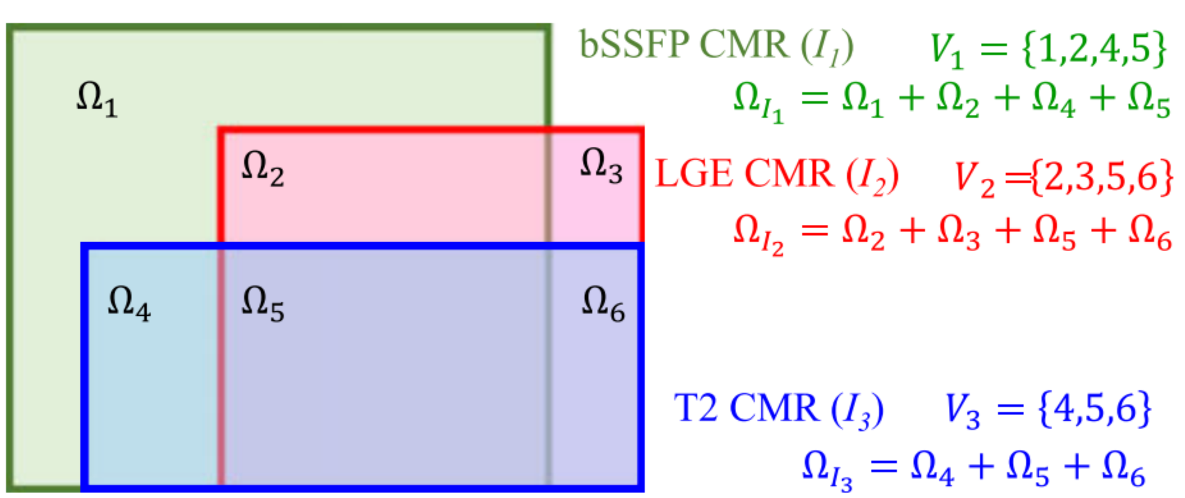


Fig. 2. Flowchart of the proposed myocardial segmentation method from the MS CMR.

Hetero-Coverage Multi-Modality Images

- In medical imaging, data from different acquisitions can generally have different resolutions and coverage of the ROI.
- The ROI of the subject is divided into N_{sr} non-overlapping sub-regions $\{\Omega_v : v = 1, \dots, N_{sr}\}$ and $\Omega = \bigcup_{v=1}^{N_{sr}} \Omega_v$.



- LL_{Ω_1} and LL_{Ω_3} use univariate mixture model ($N_v = 1$)
- LL_{Ω_2}, LL_{Ω_4} and LL_{Ω_6} use bivariate mixture model ($N_v = 2$)
- LL_{Ω_5} uses multivariate mixture model ($N_v = 3$)

Hetero-Coverage Multi-Modality Images

大数据学院 School of Data Science

- **Hetero-coverage log-likelihood:**

$$\ell(\boldsymbol{I}|\theta) = \sum_{v=1}^{N_{sr}} \sum_{x \in \Omega_v} \ln \left\{ \sum_{k \in K} \pi_{kx} \prod_{i=1}^{N_I} \sum_{c \in C_{ik}} \tau_{ikc} \cdot \Phi_{ikc}\big(I_i(x)\big) \right\}$$

- The optimization of the segmentation parameters is similar to the case of congruent data:

$$P_{kx}^{[m+1]} = \frac{p\big(\boldsymbol{I}_v(x)\big|k_x;\theta^{[m]}\big)\pi_{kx}^{[m]}}{\sum_{l \in K} p\big(\boldsymbol{I}_v(x)\big|l_x;\theta^{[m]}\big)\pi_{lx}^{[m]}}$$

$$\tau_{ikc}^{[m+1]} = \frac{\sum_{x \in \Omega_{I_i}} P_{ikcx}^{[m+1]}}{\sum_{x \in \Omega_{I_i}} \sum_{c \in C_{ik}} P_{ikcx}^{[m+1]}}$$

$$\mu_{ikc}^{[m+1]} = \frac{\sum_{x \in \Omega_{I_i}} I_i(x) P_{ikcx}^{[m+1]}}{\sum_{x \in \Omega_{I_i}} P_{ikcx}^{[m+1]}}$$

$$\left(\sigma_{ikc}^2\right)^{[m+1]} = \frac{\sum_{x \in \Omega_{I_i}} \left(I_i(x) - \mu_{ikc}^{[m+1]}\right)^2 P_{ikcx}^{[m+1]}}{\sum_{x \in \Omega_{I_i}} P_{ikcx}^{[m+1]}}$$

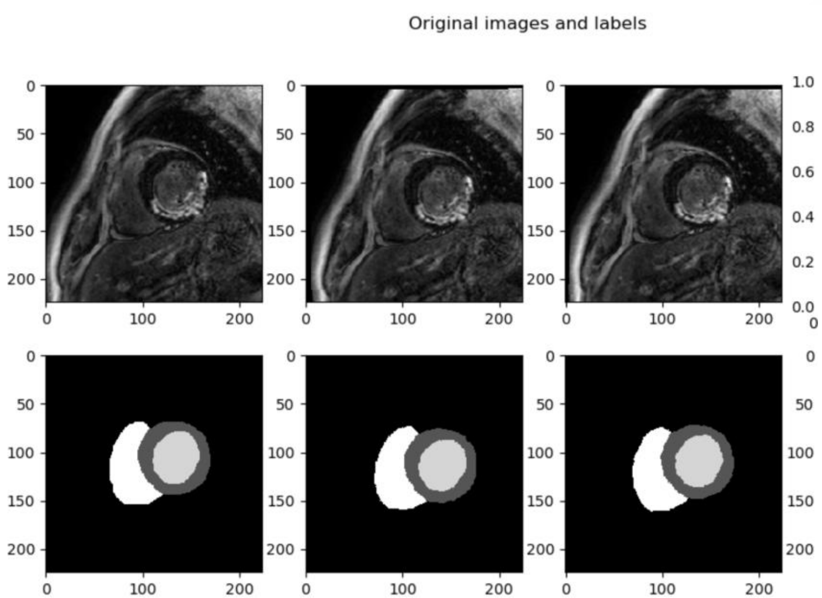
Contents

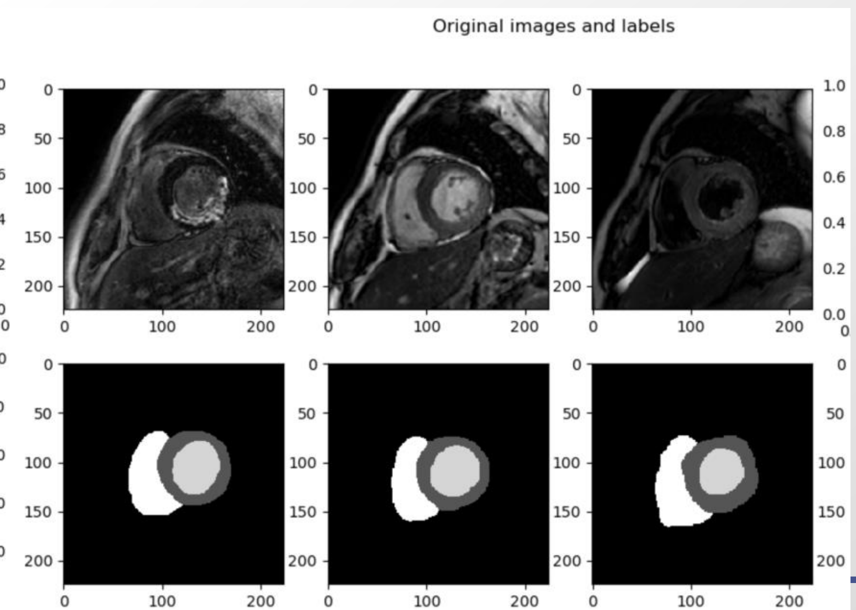
大数据学院
School of Data Science

Experiment and demonstration

大数据学院
School of Data Science

- **Data:** MSCMR dataset
- **Experimental setup:**
  - Mono-modality synthetic data from the same image slice
    - ‘patient1_DE_image_slice12’
  - Intra-patient multi-modality image slices
    - ‘patient1_DE_image_slice12’, ‘patient1_C0_image_slice5’, ‘patient1_T2_image_slice3’





Experiment and demonstration

大数据学院 School of Data Science

Initialization

```python
def init_parameters(self, images, prior, device='cpu'):
    """

    :param images: tensor of shape [1, num_subjects, 1, *vol_shape]
    :param prior: tensor of shape [1, num_classes, *vol_shape]
    :param device:
    :return:
    """
    self.mask = self._spatial_filter(prior[:, 1:].sum(dim=1, keepdim=True),
                                     image_utils.gauss_kernel1d(self.mask_sigma)).gt(self.eps).to(torch.float32)

    self.pi = torch.sum(prior * self.mask, dim=list(range(2, 2 + self.dimension)),
                        keepdim=True) / torch.sum(prior * self.mask, dim=list(range(1, 2 + self.dimension)), keepdim=True)
    self.prior = utils.compute_normalized_prob(self.pi * prior, dim=1)

    self.tau = [torch.full((self.num_subjects, self.num_subtypes[i]), 1 / self.num_subtypes[i])
                for i in range(self.num_classes)]

    mu_k = torch.sum(images * prior.unsqueeze(1) * self.mask,
                     dim=list(range(3, 3 + self.dimension))) / torch.sum(prior.unsqueeze(1) * self.mask,
                                                                        dim=list(range(3, 3 + self.dimension))).clamp(min=self
    sigma2_k = torch.sum((images - mu_k.view(1, -1, self.num_classes, *[1] * self.dimension)) ** 2 * prior.unsqueeze(1) * self.
                         dim=list(range(3, 3 + self.dimension))) / torch.sum(prior.unsqueeze(1) * self.mask,
                                                                            dim=list(range(3, 3 + self.dimension))).clamp(min=s

    self.mu = []
    for i in range(self.num_classes):
        if self.num_subtypes[i] == 1:
            self.mu.append(mu_k[:, :, [i]].squeeze(0))  # [num_subjects, 1]
        else:
            a = torch.linspace(-1, 1, steps=self.num_subtypes[i])  # [num_subtypes[i]]
            self.mu.append(mu_k[:, :, [i]].squeeze(0) + a.unsqueeze(0) * sigma2_k[:, :, [i]].squeeze(0).sqrt())  # [num_subjects
```

Experiment and demonstration

大数据学院
School of Data Science

Initialization

```
self.tau = [torch.full((self.num_subjects, self.num_subtypes[i]), 1 / self.num_subtypes[i])
            for i in range(self.num_classes)]

mu_k = torch.sum(images * prior.unsqueeze(1) * self.mask,
                 dim=list(range(3, 3 + self.dimension))) / torch.sum(prior.unsqueeze(1) * self.mask,
                                                                     dim=list(range(3, 3 + self.dimension))).clamp(min=self.
sigma2_k = torch.sum((images - mu_k.view(1, -1, self.num_classes, *[1] * self.dimension)) ** 2 * prior.unsqueeze(1) * self.m
                     dim=list(range(3, 3 + self.dimension))) / torch.sum(prior.unsqueeze(1) * self.mask,
                                                                         dim=list(range(3, 3 + self.dimension))).clamp(min=s

self.mu = []
for i in range(self.num_classes):
    if self.num_subtypes[i] == 1:
        self.mu.append(mu_k[:, :, [i]].squeeze(0))  # [num_subjects, 1]
    else:
        a = torch.linspace(-1, 1, steps=self.num_subtypes[i])  # [num_subtypes[i]]
        self.mu.append(mu_k[:, :, [i]].squeeze(0) + a.unsqueeze(0) * sigma2_k[:, :, [i]].squeeze(0).sqrt())  # [num_subjects

self.sigma2 = [sigma2_k[:, :, [i]].squeeze(0).mul(self.num_subtypes[i]).repeat(1, self.num_subtypes[i])
               for i in range(self.num_classes)]

self.posterior = self.prior

# to device
self.pi = self.pi.to(device)
self.prior = self.prior.to(device)
self.posterior = self.posterior.to(device)
for i in range(self.num_classes):
    self.tau[i] = self.tau[i].to(device)
    self.mu[i] = self.mu[i].to(device)
    self.sigma2[i] = self.sigma2[i].to(device)
```

Experiment and demonstration

大数据学院
School of Data Science

Forward

```python
    def forward(self, images, prior, **kwargs):
        """

        :param images: tensor of shape [1, num_subjects, img_channels, *vol_shape]
        :param prior: tensor of shape [1, num_classes, *vol_shape]
        :return:
        """
        flow_scale = sorted(kwargs.pop('flow_scale', (0,)), reverse=True)
        # freeze gradient
        for s in self.flow_scales:
            if s in flow_scale:
                self.vectors['scale_%s' % s].requires_grad = True
            else:
                self.vectors['scale_%s' % s].requires_grad = False

        if self.dimension == 2:
            upsample_mode = 'bilinear'
        elif self.dimension == 3:
            upsample_mode = 'trilinear'
        else:
            raise NotImplementedError

        # registration
        self.flows = [F.upsample(self.vectors['scale_%s' % s], scale_factor=2 ** s, mode=upsample_mode,
                                 align_corners=True) for s in self.flow_scales]
        if self.transform == 'rigid':
            self.theta = [torch.stack([torch.cat([torch.cos(self.rotate_params[i]),
                                                  - torch.sin(self.rotate_params[i]),
                                                  self.transl_params[i][0]]),
                                       torch.cat([torch.sin(self.rotate_params[i]),
                                                  torch.cos(self.rotate_params[i]),
```

Experiment and demonstration

大数据学院 School of Data Science

Forward

```python
for s in self.flow_scales:
    if s in flow_scale:
        self.vectors['scale_%s' % s].requires_grad = True
    else:
        self.vectors['scale_%s' % s].requires_grad = False

if self.dimension == 2:
    upsample_mode = 'bilinear'
elif self.dimension == 3:
    upsample_mode = 'trilinear'
else:
    raise NotImplementedError

# registration
self.flows = [F.upsample(self.vectors['scale_%s' % s], scale_factor=2 ** s, mode=upsample_mode,
                         align_corners=True) for s in self.flow_scales]
if self.transform == 'rigid':
    self.theta = [torch.stack([torch.cat([torch.cos(self.rotate_params[i]),
                                          - torch.sin(self.rotate_params[i]),
                                          self.transl_params[i][0]]),
                               torch.cat([torch.sin(self.rotate_params[i]),
                                          torch.cos(self.rotate_params[i]),
                                          self.transl_params[i][1]])]).unsqueeze(0)
                  for i in range(self.num_subjects - 1)]

reg_mode = kwargs.pop('reg_mode', None)
warped_prior = self.transform_prior(prior)
self.warped_mask = self.transform_prior(self.mask, mode='nearest').detach()
warped_prior = utils.compute_normalized_prob(warped_prior * self.pi)
warped_images = torch.stack(self.transform_images(images, mode=reg_mode), dim=1)

return warped_images, warped_prior
```

Experiment and demonstration

大数据学院
School of Data Science

EM update

```python
def update(self, warped_images_grad, warped_prior_grad):
    """
    update appearance parameters

    :param warped_images: tensor of shape [1, num_subjects, 1, *vol_shape]
    :param warped_prior: tensor of shape [1, num_classes, *vol_shape]
    :return:
    """

    # E-step: update the posterior
    class_cpds_grad, subtype_cpds_grad = self.compute_subtype_class_cpds(warped_images_grad)

    # detach variables from gradient calculation
    subtype_cpds = [[cpd.detach() for cpd in subtype_cpds_grad[i]] for i in range(self.num_subjects)]
    class_cpds = [cpd.detach() for cpd in class_cpds_grad]
    warped_images = warped_images_grad.detach()
    warped_prior = warped_prior_grad.detach()

    data_likelihood = torch.stack(class_cpds, dim=1).clamp(min=self.eps).log().sum(dim=1).exp()
    self.posterior = utils.compute_normalized_prob(data_likelihood * warped_prior, dim=1)  # [1, num_classes, *vol_shape]

    # M-step: update the parameters
    # update pi, prior
    self.pi = torch.sum(self.posterior * self.warped_mask, dim=list(range(2, 2 + self.dimension)),
                        keepdim=True) / torch.sum(warped_prior / torch.sum(warped_prior * self.pi, dim=1,
                                                                             keepdim=True).clamp(min=self.eps) * self.warped_mask,
                                                               dim=list(range(2, 2 + self.dimension)), keepdim=True).clamp(min=self.eps)
    # warped_prior_grad = utils.compute_normalized_prob(self.pi * warped_prior_grad, dim=1)
    # print(self.pi.requires_grad, self.prior.requires_grad)

    # update tau, mu, sigma2
    for i in range(self.num_classes):
```

Experiment and demonstration

大数据学院
School of Data Science

EM update

```
    # print(self.pi.requires_grad, self.prior.requires_grad)

    # update tau, mu, sigma2
    for i in range(self.num_classes):
        tau_ = utils.compute_normalized_prob(
            self.tau[i].view(1, self.num_subjects, self.num_subtypes[i],
                             *[1] * self.dimension) * torch.stack([cpd[i] for cpd in subtype_cpds], dim=1),
            dim=2) * self.posterior[:, [i]].unsqueeze(1)  # [1, num_subjects, num_subtypes[i], *vol_shape]
        # print(tau_.requires_grad)
        tau_ = tau_ * self.warped_mask.unsqueeze(1)

        self.tau[i] = utils.compute_normalized_prob(
            tau_.sum(dim=(0, *[i + 3 for i in range(self.dimension)])),
            dim=1)  # update tau, [num_subjects, num_subtypes[i]]
        # print(self.tau[i].requires_grad)

        self.mu[i] = torch.sum(
            tau_ * warped_images, dim=(0, *[i + 3 for i in range(self.dimension)])) / tau_.sum(
            dim=(0, *[i + 3 for i in range(self.dimension)])
        ).clamp(min=self.eps)  # update mu, [num_subjects, num_subtypes[i]]
        # print(self.mu[i].requires_grad)

        self.sigma2[i] = torch.sum(
            tau_ * (warped_images - self.mu[i].view(1, self.num_subjects, self.num_subtypes[i],
                                                     *[1] * self.dimension)
                    ) ** 2, dim=(0, *[i + 3 for i in range(self.dimension)])) / tau_.sum(
            dim=(0, *[i + 3 for i in range(self.dimension)])
        ).clamp(min=self.eps)  # update sigma square, [num_subjects, num_subtypes[i]]
        # print(self.sigma2[i].requires_grad)

    return class_cpds_grad
```

Experiment and demonstration

大数据学院
School of Data Science

Likelihood function

```python
        return class_cpds_grad

    def _compute_data_likelihood(self, class_cpds_grad):
        """

        :param class_cpds_grad: list of conditional probabilistic distributions, each of shape [batch, num_class, *vol_shape]
        :return: tensor of shape [batch, num_classes, *vol_shape]
        """
        data_likelihood = torch.stack(class_cpds_grad, dim=1).clamp(min=self.eps).log().sum(dim=1).exp()

        return data_likelihood

    def estimate_posterior(self, *args, **kwargs):
        # likelihood = self._compute_likelihood(*args, **kwargs)
        return self.posterior

    def loss_function(self, class_cpds_grad, warped_prior_grad, alpha=0.1, beta=0, gamma=0, **kwargs):
        likelihood = self._compute_data_likelihood(class_cpds_grad) * warped_prior_grad

        likelihood = likelihood * self.warped_mask

        sum_likelihood = likelihood.sum(dim=1)
        log_likelihood = sum_likelihood.clamp_min(self.eps).log()

        # print(mask.sum().item())
        mask = (likelihood > self.eps).to(torch.float32)
        loss = - torch.sum(log_likelihood * mask) / torch.sum(mask).add(self.eps)
        regularization = self._compute_regularization(alpha, beta, gamma)
        # print(loss.item(), regularization.item())
        loss += regularization

        return loss
```

Experiment and demonstration

Data preprocessing:

```
print(os.getcwd())

# load data
images = [image_utils.load_image_nii(name)[0] for name in args.image_names]  # [224, 224, 1]
labels = [image_utils.load_image_nii(name.replace('image', 'label'))[0] for name in args.image_names]

atlas_label = image_utils.load_image_nii(args.atlas_name)[0]
original_atlas = torch.from_numpy(atlas_label).unsqueeze(0).unsqueeze(0).squeeze(-1)

# preprocess data
images = [image.squeeze(2) for image in images]
original_images = torch.stack([torch.from_numpy(image) for image in images]).unsqueeze(1).unsqueeze(0)  # [1, 3, 1, 224, 224]
labels = [label.squeeze(2) for label in labels]
original_labels = torch.stack([torch.from_numpy(label) for label in labels]).unsqueeze(1).unsqueeze(0)

images = [np.clip(image, np.percentile(image, 1), np.percentile(image, 99)) for image in images]
images = [image_utils.normalize_image(image, normalization='min-max') for image in images]
labels = [image_utils.get_one_hot_label(label, args.label_intensities, channel_first=True) for label in labels]
atlas_label = image_utils.get_one_hot_label(atlas_label.squeeze(2), args.label_intensities, channel_first=True)


fig, ax = plt.subplots(4, 3, figsize=(12, 10))

# transfer data as input
images = torch.from_numpy(np.stack(images)).unsqueeze(1).unsqueeze(0)  # [1, 3, 1, 224, 224]
images = torch.stack([image_utils.separable_filter2d(images[:, i], image_utils.gauss_kernel1d(1)) for i in range(3)], dim=1)
labels = torch.from_numpy(np.stack(labels)).unsqueeze(0)  # [1, 3, 4, 224, 224]
atlas_label = torch.from_numpy(atlas_label).unsqueeze(0)  # [1, 4, 224, 224]
prior = image_utils.get_prob_from_label(atlas_label, dimension=2, sigma=2)

# set metric
Dice = metrics.OverlapMetrics(type='average_foreground_dice')
```

Experiment and demonstration

大数据学院
School of Data Science

Optimization

```python
plt.show()

for step in range(training_iters):
    optimizer.zero_grad()

    # update segmentation parameters
    warped_images, warped_prior = model(images, prior=prior, flow_scales=(0, 1, 2 ))

    for j in range(EM_steps):
        _ = model.update(warped_images.detach(), warped_prior.detach())

    # update registration parameters
    class_cpds_grad, _ = model.compute_subtype_class_cpds(warped_images)

    loss = model.loss_function(class_cpds_grad, warped_prior, alpha=bending_energy)

    loss.backward()

    optimizer.step()

    if step % display_steps == (display_steps - 1):
        print(model.pi.squeeze(), model.pi.sum())
        dice = []
        warped_labels = model.transform_labels(labels)
        for i in range(model.num_subjects - 1):
            dice.append(Dice(labels[:, 0], warped_labels[i + 1]).mean().item())
        warped_atlas_label = model.transform_prior(prior=atlas_label, mode='nearest')
        atlas_dice = Dice(labels[:, 0], warped_atlas_label).mean().item()

        print("[Validation] Step: %s, Loss: %.4f, Dice: %.4f, Atlas Dice: %.4f" % (step, loss.item(),
                                                                                   np.mean(dice), atlas_dice))
```

Contents

Conclusion

大数据学院
School of Data Science

- **Probabilistic graphical model** integrates inference and learning, which combines segmentation and registration in a unified framework.
- **Generative modelling** facilitates combined computing from multivariate images by presuming suitable conditional independences and designating proper data generating distributions.
- **Further reading:**

  - Zhuang, Xiahai. "Multivariate mixture model for myocardial segmentation combining multi-source images." *IEEE transactions on pattern analysis and machine intelligence* 41, no. 12 (2019): 2933-2946.
  - Blaiotta, Claudia, Patrick Freund, M. Jorge Cardoso, and John Ashburner. "Generative diffeomorphic modelling of large MRI data sets for probabilistic template construction." *NeuroImage* 166 (2018): 117-134.
  - Bishop, Christopher M. *Pattern recognition and machine learning*. springer, 2006.
  - Koller, Daphne, and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.





Thank You！