Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий Інститут прикладного системного аналізу

КУРСОВА РОБОТА

з дисципліни:

«Методи та технології напівкерованого навчання»

за темою:

« Модель для напівкерованої 3D сегментації лівого передсердя »

**Виконав:**

**студент групи КІ-31мп**

**Поліщук М. С.**

**Прийняв:**

**Синєглазов В.М.**

**Київ — 2024**

# Реферат

Навчання глибоких згорткових нейронних мереж зазвичай вимагає великої кількості мічених даних. Проте анотування даних для завдань сегментації медичних зображень є дорогим і трудомістким процесом. У цій роботі ми представляємо нову невизначенісно-орієнтовану напівкеровану структуру для сегментації лівого передсердя на 3D MP-зображеннях. Наша структура ефективно використовує немічені дані, заохочуючи узгоджені прогнози одного і того ж вхідного сигналу при різних збуреннях. Конкретно, структура складається з моделі студента та моделі вчителя, і модель студента навчається від моделі вчителя, мінімізуючи втрати сегментації та втрати узгодженості щодо цілей моделі вчителя. Ми розробили нову схему, орієнтовану на невизначеність, яка дозволяє моделі студента поступово навчатися з осмислених і надійних цілей, використовуючи інформацію про невизначеність. Експерименти показують, що наш метод досягає високих результатів завдяки включенню немічених даних. Наш метод перевершує передові напівкеровані методи, демонструючи потенціал нашої структури для вирішення складних завдань напівкерованого навчання.

Зміст

# ВСТУП

Автоматизована сегментація лівого передсердя (LA) на магнітно-резонансних (MR) зображеннях має велике значення для покращення лікування фібриляції передсердь. Завдяки великій кількості розмічених даних, глибоке навчання значно покращило сегментацію LA. Проте в галузі медичної візуалізації створення надійних анотацій з 3D медичних зображень у режимі зріз-за-зрізом досвідченими експертами є дорогим і трудомістким процесом. Оскільки нерозмічені дані зазвичай є у великій кількості, ми зосередилися на вивченні напівконтрольованого підходу до сегментації LA, використовуючи як обмежені розмічені дані, так і великі обсяги нерозмічених даних.

Значні зусилля були докладені для використання нерозмічених даних для покращення продуктивності сегментації в медичній спільноті. Наприклад, Bai et al. [1] запропонували метод самонавчання для сегментації серцевих MR зображень, де параметри мережі та сегментація для нерозмічених даних оновлювалися по черзі. Крім того, у напівкерованому навчанні використовувалося змагальне навчання. Zhang et al. [18] розробили глибоку змагальну мережу для використання неанотованих зображень, заохочуючи сегментацію неанотованих зображень до подібності з анотованими. Інший підхід [12] використовував змагальну мережу для вибору надійних областей нерозмічених даних для навчання сегментаційної мережі. З огляду на перспективні результати, досягнуті методами самонавчання у напівконтрольованій класифікації природних зображень, Li et al. [10] розширили модель-усереднювач для напівконтрольованої сегментації шкірних уражень. Інші підходи використовували цільову узгодженість зважених середніх для напівконтрольованої MR сегментації. Хоча було досягнуто значного прогресу, ці методи не враховують надійність цілей, що може призвести до беззмістовного керування.

У цій роботі ми представляємо нову структуру напівкерованого навчання з урахуванням невизначеності для сегментації лівого передсердя з 3D MR зображень, додатково використовуючи нерозмічені дані. Наш метод заохочує консистентність сегментаційних передбачень при різних збуреннях для одного й того ж введення, дотримуючись тієї ж ідеї, що й mean teacher. Зокрема, ми створюємо модель-вчитель і модель-учень, де модель-учень навчається від моделі-вчителя, мінімізуючи сегментаційну втрату на розмічених даних і втрату консистентності щодо цілей від моделі-вчителя на всіх вхідних даних. Без наданих істинних значень у нерозмічених даних, передбачена мета від моделі-вчителя може бути ненадійною та шумною. У цьому відношенні ми розробляємо структуру mean teacher з урахуванням невизначеності (UA-MT), де модель-учень поступово навчається від змістовних і надійних цілей, використовуючи інформацію про невизначеність моделі-вчителя. Конкретно, крім генерації цільових виходів, модель-вчитель також оцінює невизначеність кожного цільового передбачення за допомогою семплінгу Монте-Карло. Завдяки керівництву оціненої невизначеності, ми відфільтровуємо ненадійні передбачення і зберігаємо лише надійні (низька невизначеність) під час розрахунку втрати консистентності. Таким чином, модель-учень оптимізується з більш надійним керівництвом і, в свою чергу, заохочує модель-вчителя генерувати високоякісніші цілі. Наш метод був всебічно оцінений на наборі даних cardiography. Результати демонструють, що наш напівконтрольований метод досягає значних покращень у сегментації LA, використовуючи нерозмічені дані, і також перевершує інші найсучасніші методи напівконтрольованої сегментації.

# Особливості моделі

На Рис. 1 зображена наша рамка навчання з урахуванням невизначеності та самооб'єднання моделі-вчителя (UA-MT) для напівкерованої сегментації лівого передсердя (LA). Модель-вчитель генерує цілі для моделі-учня для навчання та також оцінює невизначеність цілей. Кероване невизначеністю втрата консистентності поліпшує модель-учень та стійкість рамки.

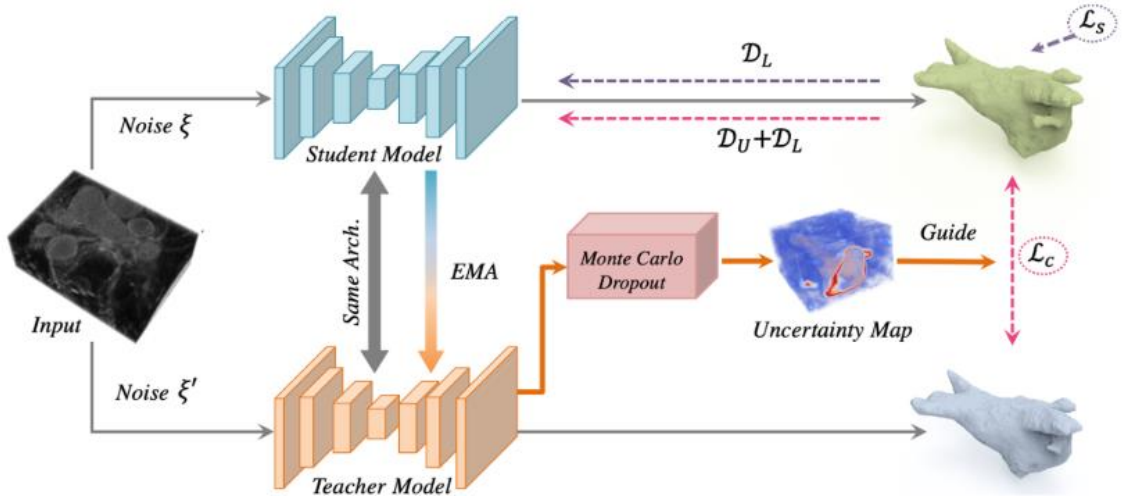


Рис 1.

**Рис. 1:** Схема нашої рамки з урахуванням невизначеності для напівкерованої сегментації. Модель-учень оптимізується шляхом мінімізації наглядної втрати $L_S$ на розмічених даних $D_L$ та втрати консистентності $L_c$ як на нерозмічених даних $D_u$, так і на розмічених даних $D_L$. Оцінена невизначеність від моделі-вчителя направляє модель-учня на вивчення надійніших цілей від моделі-вчителя.

У нашому дослідженні ми вивчаємо задачу напівкерованої сегментації для тривимірних даних, де навчальний набір складається з N мічених даних і M немічених даних. Ми позначаємо $\mathcal{D}_L = \{(x_i, y_i)\}_{i=1}^{N}$ мічений набір як і $\mathcal{D}_U = \{x_i\}_{i=N+1}^{N+M}$ немічений набір як , де $x_i \in \mathbb{R}^{H\times W\times D}$ є вхідним об'ємом, а $y_i \in \{0,1\}^{H\times W\times D}$ є еталонними анотаціями. Метою нашої напівкерованої сегментаційної структури є мінімізація наступної комбінованої цільової функції:

$$\min_{\theta} \sum_{i=1}^{N} \mathcal{L}_s(f(x_i;\theta), y_i) + \lambda \sum_{i=1}^{N+M} \mathcal{L}_c(f(x_i;\theta',\xi'), f(x_i;\theta,\xi)),$$

де Ls позначає втрати за навчанням (наприклад, втрати крос-ентропії) для оцінки якості виходу мережі на мічених вхідних даних, а Lc представляє втрати невизначеності для вимірювання узгодженості між прогнозом моделі вчителя і моделі студента для одного й того ж вхідного $x_i$ під різними збуреннями. Тут функція f(·)позначає сегментаційну нейронну мережу; $(\theta',\xi')$ $(\theta,\xi)$ представляють ваги та різні операції збурення (наприклад, додавання шуму до входу та випадкові вимикання в мережі) моделей вчителя і студента відповідно. Λ є коефіцієнтом зважування для поступового налаштування, який контролює баланс між втратами за навчанням і втратами невизначеності.

Останні дослідження [9, 14] показують, що ансамблювання прогнозів мережі на різних етапах навчання може покращити якість прогнозів, а використання їх у якості прогнозів вчителя може покращити результати. Тому ми оновлюємо ваги вчителя θ’ як експоненційний зміщений середній (EMA) ваг студента θ, щоб занести інформацію з різних етапів навчання [14];

Конкретно, ми оновлюємо ваги вчителя θ на кроці навчання t’ за формулою:

$\theta'_t = \alpha\theta'_{t-1} + (1-\alpha)\theta_t$, де α - це коефіцієнт згладжування EMA, який контролює швидкість оновлення.

**Framework "Mean Teacher", який враховує невизначеність**

Без анотацій у немічених вхідних даних передбачувані цілі від моделі вчителя можуть бути ненадійними та зашумленими. Тому ми розробляємо схему, орієнтовану на невизначеність, щоб дозволити моделі студента поступово навчатися від більш надійних цілей. Для даного батчу навчальних зображень модель вчителя не лише генерує цільові прогнози, але й оцінює невизначеність для кожної цілі. Потім модель студента оптимізується за допомогою втрат узгодженості, які зосереджені лише на впевнених цілях під керівництвом оціненої невизначеності.

Оцінка невизначеності

Мотивовані оцінкою невизначеності в баєсових мережах, ми оцінюємо невизначеність за допомогою методу Monte Carlo Dropout [8]. Детальніше, ми виконуємо T стохастичних проходів вперед через модель вчителя під випадковими відключеннями та додаванням гауссового шуму до вхідних даних. Таким чином, для кожного вокселя у вхідному об'ємі ми отримуємо набір векторів ймовірностей softmax: $\{\mathrm{p}_t\}_{t=1}^{T}$. Ми обираємо предиктивну ентропію як метрику для наближення невизначеності, оскільки вона має фіксований діапазон [8]. Формально, предиктивну ентропію можна підсумувати так:

$$\mu_c = \frac{1}{T}\sum_t \mathrm{p}_t^c \qquad \text{and} \qquad u = -\sum_c \mu_c \log \mu_c$$

де $\mathrm{p}_{ct}$ є ймовірністю e-го класу у t-му прогнозі. Зверніть увагу, що невизначеність оцінюється на рівні вокселів, і невизначеність усього об'єму U становить $\{\mathrm{u}\} \in \mathrm{R}^{\mathrm{H}\times\mathrm{W}\times\mathrm{D}}$.

**Втрати узгодженості з урахуванням невизначеності**

Під керівництвом оціненої невизначеності U ми відфільтровуємо відносно ненадійні (з високою невизначеністю) прогнози та обираємо лише певні прогнози як цілі для навчання моделі студента. Зокрема, для нашого завдання напівконтрольованої сегментації ми розробляємо втрати узгодженості з урахуванням невизначеності $L_c$ як середньоквадратичну похибку (MSE) на рівні вокселів між моделями вчителя і студента лише для найбільш впевнених прогнозів:

$$\mathcal{L}_c(f', f) = \frac{\sum_v \mathbb{I}(u_v < H) \|f'_v - f_v\|^2}{\sum_v \mathbb{I}(u_v < H)}$$

де I(·) є індикаторною функцією; $f'_v$ and $f_v$ - це прогнози моделі вчителя і моделі студента відповідно на v-му вокселі; $u_v$ - це оцінена невизначеність U на v-му вокселі; і H - це поріг для вибору найбільш впевнених цілей. Завдяки нашим втратам узгодженості з урахуванням невизначеності в процесі навчання, як студент, так і вчитель можуть навчатися більш надійним знанням, що, у свою чергу, може зменшити загальну невизначеність моделі.

**Технічні особливості**

Ми використовуємо V-Net [11] як основну архітектуру мережі. Ми прибираємо коротке резидуальне з'єднання в кожному блоку згортки та використовуємо комбіновані втрати крос-ентропії та Dice втрати [16]. Щоб адаптувати V-Net як баєсову мережу для оцінки невизначеності, два шари dropout з рівнем dropout 0.5 додаються після шарів L-Stage 5 і R-Stage 1 у V-Net. Ми включаємо dropout під час навчання мережі та оцінки невизначеності, але вимикаємо його на етапі тестування, оскільки нам не потрібно оцінювати невизначеність. Ми емпірично встановили коефіцієнт згладжування EMA α\alphaα як 0.99, посилаючись на попередню роботу [14]. Відповідно до [9,14], ми використовуємо часозалежну функцію Gaussian warming up

$\lambda(t) = 0.1 * e^{(-5(1-t/t_{max})^2)}$ для контролю балансу між втратами за навчанням та втратами узгодженості, де t позначає поточний крок навчання, а $t_{max}$ є максимальним кроком навчання. Такий дизайн забезпечує, що на початку цільові втрати переважно складаються з навчальних втрат і дозволяє уникнути ситуації, коли мережа застрягає в виродженому рішенні, де не отримуються значущі прогнози для немічених даних [9].

Для оцінки невизначеності ми встановлюємо T=8 для балансу між якістю оцінки невизначеності та ефективністю навчання. Ми також використовуємо ту ж Gaussian ramp-up парадигму для поступового збільшення порогу невизначеності H з $\frac{3}{4}U_{max}$ to $U_{max}$ є максимальним значенням невизначеності (тобто ln2 у наших експериментах). У міру продовження навчання, наш метод буде поступово фільтрувати менше даних і дозволяти моделі студента поступово навчатися від відносно впевнених до невизначених випадків.

## Навчальна вибірка

Набір даних використаний для навчання представляє собою 668 анотованих знімків ехокардіографії серця та 74 немаркованих знімки. Приклад зображень на рисунку 2.

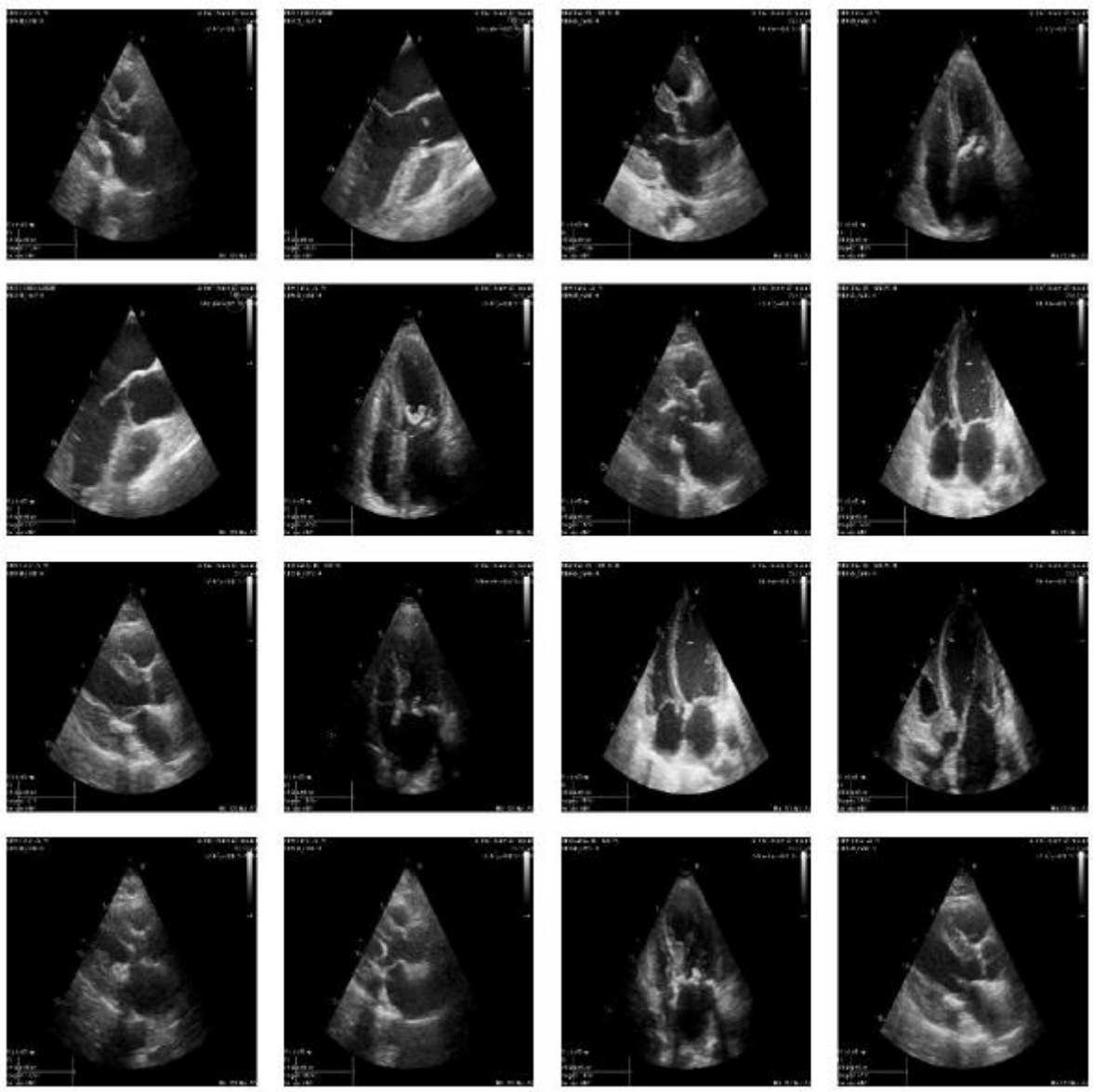

Рис.2. Зображення ехокардіографій серця з вибірки

Кожному анотованому зображенню відповідає маска, кожен піксель якої:

• 1, якщо належить до об'єкта інтересу(вегетація серця)

• 0, якщо піксель фону

Нам потрібно навчити модель таким чином, щоб вона робила передбачення бінарної маски вегетації серця кожному зображенню.

4.2. Особливості навчання

Структура була реалізована в PyTorch з використанням GPU. Ми використовували оптимізатор SGD для оновлення параметрів мережі (розпад ваг = 0.0001, імпульс = 0.9). Початкова швидкість навчання була встановлена на рівні 0.01 і зменшувалася в 10 разів кожні 2500 ітерацій. Ми проводили навчання протягом 6000 ітерацій, оскільки мережа досягла збіжності. Використаний для дослідження набір даних представляє собою 668 кадрів з кардіографа з зареєстрованою спеціалістами аномалією (вегетацією) та 668 «чистих» кадрів, загалом 1336 кадрів. Зображення були вилучені з ехокардіограм в форматі DICOM. Набір даних був наданий Інститутом серця МОЗ України.

Початковий розмір зображень відповідно був 708 на 1016 пікселів, згодом оброблений до зображень розміром 600 на 600 пікселів центральним обрізанням. Аугментації не використовувались, з огляду на те, що знімки проводять за однакових умов.

## Опис метрик для оцінювання

### Dice Similarity Coefficient

Dice Similarity Coefficient (DSC), також відомий як Dice Coefficient або F1-Score, є метрикою, що вимірює схожість між двома наборами даних, тобто виконує порівняння передбачених та реальних пікселів.

$$\mathrm{DSC} = \frac{2 \cdot |\mathrm{A} \cap \mathrm{B}|}{|A| + |B|}$$

де:

- • $A-$ - множина пікселів у передбаченій масці,
- • $B-$ множина пікселів у реальній масці,
- • $|\mathrm{A} \cap \mathrm{B}|-$ кількість спільних пікселів між передбаченою та реальною масками.

## Intersection over Union

Intersection over Union (IoU), або також відомий як Jaccard Index, є мірою збігу між двома множинами. Він використовується для оцінки точності сегментації шляхом порівняння площі перетину та об'єднання передбачених і справжніх пікселів.

$$\mathrm{IoU} = \frac{|\mathrm{A} \cap \mathrm{B}|}{|\mathrm{A} \cup \mathrm{B}|}$$

де:

- • $A$ - множина пікселів у передбаченій масці,
- • $B$ - множина пікселів у реальній масці,
- • $|\mathrm{A} \cup \mathrm{B}|$ - кількість пікселів у об'єднаній множині (передбачені або реальні пікселі).

## Precision

Precision вимірює, наскільки багато передбачених позитивних зразків є дійсно позитивними. Ця метрика важлива в контекстах, де важливо оцінити кількість хибних позитивних передбачень.

$$\text{Precision} = \frac{\text{TP} + \text{FP}}{\text{TP}}$$

де:

- • $TP$ (True Positives) - кількість правильних передбачень.
- • $FP$ (False Positives) - кількість хибних передбачень, коли модель передбачила позитивний клас, але насправді він був негативним

## Recall

Recall, вимірює здатність моделі виявляти всі реальні позитивні зразки. Ця метрика важлива в контекстах, де критично виявити всі можливі випадки позитивного класу. $Recall=TPTP+FN$

$$Recall = \frac{TP}{TP + FN}$$

де:

• TP (True Positives) - кількість правильних передбачень.

• FN (False Negatives) - кількість пропущених позитивних передбачень, коли модель не виявила позитивний клас, хоча він був присутній.

## Результати роботи

| Метод | DSC | IoU | Precision | Recall |
|---|---|---|---|---|
| Self-training | 0,8692 | 0,7721 | 0,8231 | 0,7981 |
| DAN | 0,8756 | 0,7828 | 0,8312 | 0,8012 |
| ASDNet | 0,8791 | 0,7882 | 0,8442 | 0,8612 |
| TCSE | 0,8810 | 0,7921 | 0,8716 | 0,8711 |
| Uncertainty-aware Self-ensembling Model for Semi-supervised 3D Left Atrium Segmentation | 0,8871 | 0,8023 | 0,8819 | 0,8798 |

Табл.1

Оцінка різних методів сегментації

Ми використовуємо чотири метрики для кількісної оцінки нашого методу, включаючи DSC, IoU, Precision, Recall. З усіх навчальних сканів ми використовуємо 20% як позначені дані, а решту сканів як непозначені дані. У Таблиці 1 представлені результати сегментації моделі V-Net, навченої на позначених та непозначених даних, і нашого напівкерованого методу на тестовому наборі даних.

Ми навчали повністю наглядну модель V-Net з усіма позначеними сканами, яку можна розглядати як верхню межу продуктивності. Як видно з результатів, наш напівкерований метод наближається до повністю наглядного. Для перевірки нашого дизайну основної мережі ми посилаємось на метод, зазначений у викликному висновку [4], який використовував багатозадачну модель U-Net для сегментації лівого передсердя. Порівняно з цим методом, наш V-Net можна вважати стандартною базовою моделлю.

# ВИСНОВОК

Ми розробили новий метод напівкерованого навчання, що враховує невизначеність для сегментації лівого передсердя на 3D зображеннях магнітного резонансу. Наш метод спонукає сегментацію бути послідовною для одного й того ж входу при різних зміщеннях, щоб використовувати непозначені дані. Більш того, ми досліджуємо невизначеність моделі для покращення якості цільових даних. Порівняння з іншими методами напівкерованого навчання підтверджують ефективність нашого методу. Майбутні роботи включають дослідження впливу різних способів оцінювання невизначеності та застосування нашого фреймворку до інших проблем напівкерованої сегментації медичних зображень.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Bai, W., Oktay, O., Sinclair, M.e.a.: Semi-supervised learning for network-based cardiac mr image segmentation. In: MICCAI. pp. 253{260 (2017)

2. Baur, C., Albarqouni, S., Navab, N.: Semi-supervised deep learning for fully convolutional networks. In: MICCAI. pp. 311{319 (2017)

3. Chartsias, A., Joyce, T., Papanastasiou, G., Semple, S., Williams, M., Newby, D., Dharmakumar, R., Tsaftaris, S.A.: Factorised spatial representation learning: application in semi-supervised myocardial segmentation. MICCAI pp. 490{498 (2018)

4. Chen, C., Bai, W., Rueckert, D.: Multi-task learning for left atrial segmentation on ge-mri. arXiv preprint arXiv:1810.13205 (2018)

5. Cui, W., Liu, Y., Li, Y., Guo, M., Li, Y., Li, X., Wang, T., Zeng, X., Ye, C.: Semi-supervised brain lesion segmentation with an adapted mean teacher model. In: IPMI. pp. 554{565 (2019)

6. Dong, N., Kamp meyer, M., Liang, X., Wang, Z., Dai, W., Xing, E.: Unsupervised domain adaptation for automatic estimation of cardiothoracic ratio. In: MICCAI. pp. 544{552 (2018)

7. Ganaye, P.A., Sdika, M., Benoit-Cattin, H.: Semi-supervised learning for segmentation under semantic constraint. In: MICCAI. pp. 595{602 (2018)

8. Kendall, A., Gal, Y.: What uncertainties do we need in bayesian deep learning for computer vision? In: NIPS. pp. 5574{5584 (2017)

9. Laine, S., Aila, T.: Temporal ensembling for semi-supervised learning. arXiv preprint (2016)

10. Li, X., Yu, L., Chen, H., Fu, C.W., Heng, P.A.: Semi-supervised skin lesion segmentation via transformation consistent self-ensembling model. BMVC (2018)

11. Milletari, F., Navab, N., Ahmadi, S.A.: V-net: Fully convolutional neural networks

for volumetric medical image segmentation. In: 3DV. pp. 565{571 (2016)

12. Nie, D., Gao, Y., Wang, L., Shen, D.: Asdnet: Attention based semi-supervised deep networks for medical image segmentation. In: MICCAI. pp. 370{378 (2018)

13. Perone, C.S., Cohen-Adad, J.: Deep semi-supervised segmentation with weight-averaged consistency targets. In: DLMIA workshop (2018)

14. Tarvainen, A., Valpola, H.: Mean teachers are better role models: Weight-averaged

consistency targets improve semi-supervised deep learning results. In: NIPS (2017)

15. Xiong, Z., Fedorov, V.V., Fu, X., Cheng, E., Macleod, R., Zhao, J.: Fully automatic

left atrium segmentation from late gadolinium enhanced magnetic resonance imaging using a dual fully convolutional neural network. TMI 38(2), 515{524 (2019)

16. Yang, X., Bian, C., Yu, L., Ni, D., Heng, P.A.: Hybrid loss guided convolutional networks for whole heart parsing. In: International Workshop on STACOM (2017)

17. Yu, L., Cheng, J.Z., Dou, Q., Yang, X., Chen, H., Qin, J., Heng, P.A.: Automatic 3d cardiovascular mr segmentation with densely-connected volumetric convnets. In: MICCAI. pp. 287{295. Springer (2017)

18. Zhang, Y., Yang, L., Chen, J., Fredericksen, M., Hughes, D.P., Chen, D.Z.: Deep adversarial networks for biomedical image segmentation utilizing unannotated images. In: MICCAI. pp. 408{416 (2017)

19. Zhou, Y.,Wang, Y., Tang, P., Bai, S., Shen, W., Fishman, E.K., Yuille, A.L.: Semi-supervised multi-organ segmentation via multi-planar co-training. arXiv preprint arXiv:1804.02586 (2018)

# Лістинг програми

```
import os
import argparse
import torch
from networks.vnet import VNet
from test_util import test_all_case

parser = argparse.ArgumentParser()
parser.add_argument('--root_path', type=str, default='../data/cardiography/', help='Name of Experiment')
parser.add_argument('--model', type=str,  default='vnet_supervisedonly_dp', help='model_name')
parser.add_argument('--gpu', type=str,  default='0', help='GPU to use')
FLAGS = parser.parse_args()

os.environ['CUDA_VISIBLE_DEVICES'] = FLAGS.gpu
snapshot_path = "../model/"+FLAGS.model+"/"
test_save_path = "../model/prediction/"+FLAGS.model+"_post/"
if not os.path.exists(test_save_path):
    os.makedirs(test_save_path)

num_classes = 2

with open(FLAGS.root_path + '/../test.list', 'r') as f:
    image_list = f.readlines()
image_list = [FLAGS.root_path +item.replace('\n', '')+"/mri_norm2.h5" for item in image_list]



def test_calculate_metric(epoch_num):
    net = VNet(n_channels=1, n_classes=num_classes, normalization='batchnorm', has_dropout=False).cuda()
    save_mode_path = os.path.join(snapshot_path, 'iter_' + str(epoch_num) + '.pth')
```

```
    net.load_state_dict(torch.load(save_mode_path))
    print("init weight from {}".format(save_mode_path))
    net.eval()

    avg_metric = test_all_case(net, image_list, num_classes=num_classes,
                               patch_size=(112, 112, 80), stride_xy=18, stride_z=4,
                               save_result=True, test_save_path=test_save_path)

    return avg_metric



if __name__ == '__main__':
    metric = test_calculate_metric(6000)
    print(metric)

import h5py
import math
import nibabel as nib
import numpy as np
from medpy import metric
import torch
import torch.nn.functional as F
from tqdm import tqdm



def test_all_case(net, image_list, num_classes, patch_size=(112, 112, 80), stride_xy=18, stride_z=4, save_result=True, test_save_path=None, preproc_fn=None):
    total_metric = 0.0
    for image_path in tqdm(image_list):
        id = image_path.split('/')[-1]
        h5f = h5py.File(image_path, 'r')
        image = h5f['image'][:]
```

```
        label = h5f['label'][:]
        if preproc_fn is not None:
            image = preproc_fn(image)
        prediction, score_map = test_single_case(net, image, stride_xy, stride_z, patch_size, num_classes=num_classes)

        if np.sum(prediction)==0:
            single_metric = (0,0,0,0)
        else:
            single_metric = calculate_metric_percase(prediction, label[:])
        total_metric += np.asarray(single_metric)

        if save_result:
            nib.save(nib.Nifti1Image(prediction.astype(np.float32), np.eye(4)), test_save_path + id + "_pred.nii.gz")
            nib.save(nib.Nifti1Image(image[:].astype(np.float32), np.eye(4)), test_save_path + id + "_img.nii.gz")
            nib.save(nib.Nifti1Image(label[:].astype(np.float32), np.eye(4)), test_save_path + id + "_gt.nii.gz")
    avg_metric = total_metric / len(image_list)
    print('average metric is {}'.format(avg_metric))

    return avg_metric


def test_single_case(net, image, stride_xy, stride_z, patch_size, num_classes=1):
    w, h, d = image.shape

    # if the size of image is less than patch_size, then padding it
    add_pad = False
    if w < patch_size[0]:
        w_pad = patch_size[0]-w
        add_pad = True
```

```
    else:
        w_pad = 0
    if h < patch_size[1]:
        h_pad = patch_size[1]-h
        add_pad = True
    else:
        h_pad = 0
    if d < patch_size[2]:
        d_pad = patch_size[2]-d
        add_pad = True
    else:
        d_pad = 0
    wl_pad, wr_pad = w_pad//2,w_pad-w_pad//2
    hl_pad, hr_pad = h_pad//2,h_pad-h_pad//2
    dl_pad, dr_pad = d_pad//2,d_pad-d_pad//2
    if add_pad:
        image = np.pad(image, [(wl_pad,wr_pad),(hl_pad,hr_pad), (dl_pad, dr_pad)], mode='constant', constant_values=0)
    ww,hh,dd = image.shape

    sx = math.ceil((ww - patch_size[0]) / stride_xy) + 1
    sy = math.ceil((hh - patch_size[1]) / stride_xy) + 1
    sz = math.ceil((dd - patch_size[2]) / stride_z) + 1
    print("{}, {}, {}".format(sx, sy, sz))
    score_map = np.zeros((num_classes, ) + image.shape).astype(np.float32)
    cnt = np.zeros(image.shape).astype(np.float32)

    for x in range(0, sx):
        xs = min(stride_xy*x, ww-patch_size[0])
        for y in range(0, sy):
            ys = min(stride_xy * y,hh-patch_size[1])
            for z in range(0, sz):
```

```
            zs = min(stride_z * z, dd-patch_size[2])
            test_patch = image[xs:xs+patch_size[0], ys:ys+patch_size[1], zs:zs+patch_size[2]]
            test_patch = np.expand_dims(np.expand_dims(test_patch,axis=0),axis=0).astype(np.float32)
            test_patch = torch.from_numpy(test_patch).cuda()
            y1 = net(test_patch)
            y = F.softmax(y1, dim=1)
            y = y.cpu().data.numpy()
            y = y[0,:,:,:,:]
            score_map[:, xs:xs+patch_size[0], ys:ys+patch_size[1], zs:zs+patch_size[2]] \
              = score_map[:, xs:xs+patch_size[0], ys:ys+patch_size[1], zs:zs+patch_size[2]] + y
            cnt[xs:xs+patch_size[0], ys:ys+patch_size[1], zs:zs+patch_size[2]] \
              = cnt[xs:xs+patch_size[0], ys:ys+patch_size[1], zs:zs+patch_size[2]] + 1
    score_map = score_map/np.expand_dims(cnt,axis=0)
    label_map = np.argmax(score_map, axis = 0)
    if add_pad:
        label_map = label_map[wl_pad:wl_pad+w,hl_pad:hl_pad+h,dl_pad:dl_pad+d]
        score_map = score_map[:,wl_pad:wl_pad+w,hl_pad:hl_pad+h,dl_pad:dl_pad+d]
    return label_map, score_map


def cal_dice(prediction, label, num=2):
    total_dice = np.zeros(num-1)
    for i in range(1, num):
        prediction_tmp = (prediction==i)
        label_tmp = (label==i)
        prediction_tmp = prediction_tmp.astype(np.float)
        label_tmp = label_tmp.astype(np.float)

        dice = 2 * np.sum(prediction_tmp * label_tmp) / (np.sum(prediction_tmp) + np.sum(label_tmp))
        total_dice[i - 1] += dice
```

```
    return total_dice


def calculate_metric_percase(pred, gt):
    dice = metric.binary.dc(pred, gt)
    jc = metric.binary.jc(pred, gt)
    hd = metric.binary.hd95(pred, gt)
    asd = metric.binary.asd(pred, gt)

    return dice, jc, hd, asd

import os
import sys
from tqdm import tqdm
from tensorboardX import SummaryWriter
import shutil
import argparse
import logging
import time
import random
import numpy as np

import torch
import torch.optim as optim
from torchvision import transforms
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
from torch.utils.data import DataLoader
from torchvision.utils import make_grid

from networks.vnet import VNet
from utils.losses import dice_loss
```

```
from dataloaders.la_heart import LAHeart, RandomCrop, CenterCrop, RandomRotFlip, ToTensor, TwoStreamBatchSampler



parser = argparse.ArgumentParser()

parser.add_argument('--root_path', type=str, default='../data/cardiography/', help='Name of Experiment')

parser.add_argument('--exp', type=str,  default='vnet_supervisedonly_dp', help='model_name')

parser.add_argument('--max_iterations', type=int,  default=6000, help='maximum epoch number to train')

parser.add_argument('--batch_size', type=int, default=4, help='batch_size per gpu')

parser.add_argument('--base_lr',  type=float,  default=0.01, help='maximum epoch number to train')

parser.add_argument('--deterministic', type=int,  default=1, help='whether use deterministic training')

parser.add_argument('--seed', type=int,  default=1337, help='random seed')

parser.add_argument('--gpu', type=str,  default='0', help='GPU to use')

args = parser.parse_args()


train_data_path = args.root_path

snapshot_path = "../model/" + args.exp + "/"


os.environ['CUDA_VISIBLE_DEVICES'] = args.gpu

batch_size = args.batch_size * len(args.gpu.split(','))

max_iterations = args.max_iterations

base_lr = args.base_lr


if args.deterministic:

    cudnn.benchmark = False

    cudnn.deterministic = True

    random.seed(args.seed)

    np.random.seed(args.seed)

    torch.manual_seed(args.seed)
```

```
    torch.cuda.manual_seed(args.seed)

patch_size = (112, 112, 80)
num_classes = 2

if __name__ == "__main__":
    ## make logger file
    if not os.path.exists(snapshot_path):
        os.makedirs(snapshot_path)
    if os.path.exists(snapshot_path + '/code'):
        shutil.rmtree(snapshot_path + '/code')
    shutil.copytree('.', snapshot_path + '/code', shutil.ignore_patterns(['.git','__pycache__']))

    logging.basicConfig(filename=snapshot_path+"/log.txt", level=logging.INFO,
                        format='[%(asctime)s.%(msecs)03d] %(message)s', datefmt='%H:%M:%S')
    logging.getLogger().addHandler(logging.StreamHandler(sys.stdout))
    logging.info(str(args))

    net = VNet(n_channels=1, n_classes=num_classes, normalization='batchnorm',
has_dropout=True)
    net = net.cuda()

    db_train = LAHeart(base_dir=train_data_path,
                       split='train',
                       num=16,
                       transform = transforms.Compose([
                         RandomRotFlip(),
                         RandomCrop(patch_size),
                         ToTensor(),
                         ]))
    db_test = LAHeart(base_dir=train_data_path,
                       split='test',
```

```
                transform = transforms.Compose([
                    CenterCrop(patch_size),
                    ToTensor()
                ]))
    def worker_init_fn(worker_id):
        random.seed(args.seed+worker_id)
    trainloader = DataLoader(db_train, batch_size=batch_size, shuffle=True,  num_workers=4, pin_memory=True, worker_init_fn=worker_init_fn)

    net.train()
    optimizer = optim.SGD(net.parameters(), lr=base_lr, momentum=0.9, weight_decay=0.0001)

    writer = SummaryWriter(snapshot_path+'/log')
    logging.info("{} itertations per epoch".format(len(trainloader)))

    iter_num = 0
    max_epoch = max_iterations//len(trainloader)+1
    lr_ = base_lr
    net.train()
    for epoch_num in tqdm(range(max_epoch), ncols=70):
        time1 = time.time()
        for i_batch, sampled_batch in enumerate(trainloader):
            time2 = time.time()
            # print('fetch data cost {}'.format(time2-time1))
            volume_batch, label_batch = sampled_batch['image'], sampled_batch['label']
            volume_batch, label_batch = volume_batch.cuda(), label_batch.cuda()
            outputs = net(volume_batch)

            loss_seg = F.cross_entropy(outputs, label_batch)
            outputs_soft = F.softmax(outputs, dim=1)
            loss_seg_dice = dice_loss(outputs_soft[:, 1, :, :, :], label_batch == 1)
            loss = 0.5*(loss_seg+loss_seg_dice)
```

```
optimizer.zero_grad()
loss.backward()
optimizer.step()

iter_num = iter_num + 1
writer.add_scalar('lr', lr_, iter_num)
writer.add_scalar('loss/loss_seg', loss_seg, iter_num)
writer.add_scalar('loss/loss_seg_dice', loss_seg_dice, iter_num)
writer.add_scalar('loss/loss', loss, iter_num)
logging.info('iteration %d : loss : %f' % (iter_num, loss.item()))
if iter_num % 50 == 0:
    image = volume_batch[0, 0:1, :, :, 20:61:10].permute(3,0,1,2).repeat(1,3,1,1)
    grid_image = make_grid(image, 5, normalize=True)
    writer.add_image('train/Image', grid_image, iter_num)

    outputs_soft = F.softmax(outputs, 1)
    image = outputs_soft[0, 1:2, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
    grid_image = make_grid(image, 5, normalize=False)
    writer.add_image('train/Predicted_label', grid_image, iter_num)

    image = label_batch[0, :, :, 20:61:10].unsqueeze(0).permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
    grid_image = make_grid(image, 5, normalize=False)
    writer.add_image('train/Groundtruth_label', grid_image, iter_num)

## change lr
if iter_num % 2500 == 0:
    lr_ = base_lr * 0.1 ** (iter_num // 2500)
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr_
if iter_num % 1000 == 0:
    save_mode_path = os.path.join(snapshot_path, 'iter_' + str(iter_num) + '.pth')
```

```
                torch.save(net.state_dict(), save_mode_path)
                logging.info("save model to {}".format(save_mode_path))

            if iter_num > max_iterations:
                break
            time1 = time.time()
        if iter_num > max_iterations:
            break
    save_mode_path = os.path.join(snapshot_path, 'iter_'+str(max_iterations+1)+'.pth')
    torch.save(net.state_dict(), save_mode_path)
    logging.info("save model to {}".format(save_mode_path))
    writer.close()
import os
import sys
from tqdm import tqdm
from tensorboardX import SummaryWriter
import shutil
import argparse
import logging
import time
import random
import numpy as np

import torch
import torch.optim as optim
from torchvision import transforms
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
from torch.utils.data import DataLoader
from torchvision.utils import make_grid

from networks.vnet import VNet
```

```
from dataloaders import utils

from utils import ramps, losses

from dataloaders.la_heart import LAHeart, RandomCrop, CenterCrop, RandomRotFlip, ToTensor, TwoStreamBatchSampler


parser = argparse.ArgumentParser()

parser.add_argument('--root_path', type=str, default='../data/cardiography/', help='Name of Experiment')

parser.add_argument('--exp', type=str,  default='UAMT', help='model_name')

parser.add_argument('--max_iterations', type=int,  default=6000, help='maximum epoch number to train')

parser.add_argument('--batch_size', type=int, default=4, help='batch_size per gpu')

parser.add_argument('--labeled_bs', type=int, default=2, help='labeled_batch_size per gpu')

parser.add_argument('--base_lr',  type=float,  default=0.01, help='maximum epoch number to train')

parser.add_argument('--deterministic', type=int,  default=1, help='whether use deterministic training')

parser.add_argument('--seed', type=int,  default=1337, help='random seed')

parser.add_argument('--gpu', type=str,  default='0', help='GPU to use')

### costs

parser.add_argument('--ema_decay', type=float,  default=0.99, help='ema_decay')

parser.add_argument('--consistency_type', type=str,  default="mse", help='consistency_type')

parser.add_argument('--consistency', type=float,  default=0.1, help='consistency')

parser.add_argument('--consistency_rampup', type=float, default=40.0, help='consistency_rampup')

args = parser.parse_args()


train_data_path = args.root_path

snapshot_path = "../model/" + args.exp + "/"



os.environ['CUDA_VISIBLE_DEVICES'] = args.gpu

batch_size = args.batch_size * len(args.gpu.split(','))

max_iterations = args.max_iterations
```

```
base_lr = args.base_lr
labeled_bs = args.labeled_bs

if args.deterministic:
    cudnn.benchmark = False
    cudnn.deterministic = True
    random.seed(args.seed)
    np.random.seed(args.seed)
    torch.manual_seed(args.seed)
    torch.cuda.manual_seed(args.seed)

num_classes = 2
patch_size = (112, 112, 80)

def get_current_consistency_weight(epoch):
    # Consistency ramp-up from https://arxiv.org/abs/1610.02242
    return args.consistency * ramps.sigmoid_rampup(epoch, args.consistency_rampup)

def update_ema_variables(model, ema_model, alpha, global_step):
    # Use the true average until the exponential average is more correct
    alpha = min(1 - 1 / (global_step + 1), alpha)
    for ema_param, param in zip(ema_model.parameters(), model.parameters()):
        ema_param.data.mul_(alpha).add_(1 - alpha, param.data)

if __name__ == "__main__":
    ## make logger file
    if not os.path.exists(snapshot_path):
        os.makedirs(snapshot_path)
    if os.path.exists(snapshot_path + '/code'):
        shutil.rmtree(snapshot_path + '/code')
    shutil.copytree('.', snapshot_path + '/code', shutil.ignore_patterns(['.git','__pycache__']))
```

```
    logging.basicConfig(filename=snapshot_path+"/log.txt", level=logging.INFO,
                        format='[%(asctime)s.%(msecs)03d] %(message)s', datefmt='%H:%M:%S')
    logging.getLogger().addHandler(logging.StreamHandler(sys.stdout))
    logging.info(str(args))


    def create_model(ema=False):
        # Network definition
        net = VNet(n_channels=1, n_classes=num_classes, normalization='batchnorm', has_dropout=True)
        model = net.cuda()
        if ema:
            for param in model.parameters():
                param.detach_()
        return model


    model = create_model()
    ema_model = create_model(ema=True)


    db_train = LAHeart(base_dir=train_data_path,
                       split='train',
                       transform = transforms.Compose([
                          RandomRotFlip(),
                          RandomCrop(patch_size),
                          ToTensor(),
                          ]))
    db_test = LAHeart(base_dir=train_data_path,
                       split='test',
                       transform = transforms.Compose([
                           CenterCrop(patch_size),
                           ToTensor()
                       ]))
    labeled_idxs = list(range(16))
```

```
    unlabeled_idxs = list(range(16, 80))
    batch_sampler = TwoStreamBatchSampler(labeled_idxs, unlabeled_idxs, batch_size, batch_size-labeled_bs)
    def worker_init_fn(worker_id):
        random.seed(args.seed+worker_id)
    trainloader = DataLoader(db_train, batch_sampler=batch_sampler, num_workers=4, pin_memory=True,worker_init_fn=worker_init_fn)


    model.train()
    ema_model.train()
    optimizer = optim.SGD(model.parameters(), lr=base_lr, momentum=0.9, weight_decay=0.0001)


    if args.consistency_type == 'mse':
        consistency_criterion = losses.softmax_mse_loss
    elif args.consistency_type == 'kl':
        consistency_criterion = losses.softmax_kl_loss
    else:
        assert False, args.consistency_type


    writer = SummaryWriter(snapshot_path+'/log')
    logging.info("{} itertations per epoch".format(len(trainloader)))


    iter_num = 0
    max_epoch = max_iterations//len(trainloader)+1
    lr_ = base_lr
    model.train()
    for epoch_num in tqdm(range(max_epoch), ncols=70):
        time1 = time.time()
        for i_batch, sampled_batch in enumerate(trainloader):
            time2 = time.time()
            # print('fetch data cost {}'.format(time2-time1))
            volume_batch, label_batch = sampled_batch['image'], sampled_batch['label']
```

```
        volume_batch, label_batch = volume_batch.cuda(), label_batch.cuda()


        noise = torch.clamp(torch.randn_like(volume_batch) * 0.1, -0.2, 0.2)
        ema_inputs = volume_batch + noise
        outputs = model(volume_batch)
        with torch.no_grad():
            ema_output = ema_model(ema_inputs)
        T = 8
        volume_batch_r = volume_batch.repeat(2, 1, 1, 1, 1)
        stride = volume_batch_r.shape[0] // 2
        preds = torch.zeros([stride * T, 2, 112, 112, 80]).cuda()
        for i in range(T//2):
            ema_inputs = volume_batch_r + torch.clamp(torch.randn_like(volume_batch_r) * 0.1, -0.2, 0.2)
            with torch.no_grad():
                preds[2 * stride * i:2 * stride * (i + 1)] = ema_model(ema_inputs)
        preds = F.softmax(preds, dim=1)
        preds = preds.reshape(T, stride, 2, 112, 112, 80)
        preds = torch.mean(preds, dim=0)  #(batch, 2, 112,112,80)
        uncertainty = -1.0*torch.sum(preds*torch.log(preds + 1e-6), dim=1, keepdim=True) #(batch, 1, 112,112,80)


        ## calculate the loss
        loss_seg = F.cross_entropy(outputs[:labeled_bs], label_batch[:labeled_bs])
        outputs_soft = F.softmax(outputs, dim=1)
        loss_seg_dice = losses.dice_loss(outputs_soft[:labeled_bs, 1, :, :, :], label_batch[:labeled_bs] == 1)


        consistency_weight = get_current_consistency_weight(iter_num//150)
        consistency_dist = consistency_criterion(outputs, ema_output) #(batch, 2, 112,112,80)
        threshold = (0.75+0.25*ramps.sigmoid_rampup(iter_num, max_iterations))*np.log(2)
        mask = (uncertainty<threshold).float()
        consistency_dist = torch.sum(mask*consistency_dist)/(2*torch.sum(mask)+1e-16)
```

```
        consistency_loss = consistency_weight * consistency_dist
        loss = 0.5*(loss_seg+loss_seg_dice) + consistency_loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        update_ema_variables(model, ema_model, args.ema_decay, iter_num)

        iter_num = iter_num + 1
        writer.add_scalar('uncertainty/mean', uncertainty[0,0].mean(), iter_num)
        writer.add_scalar('uncertainty/max', uncertainty[0,0].max(), iter_num)
        writer.add_scalar('uncertainty/min', uncertainty[0,0].min(), iter_num)
        writer.add_scalar('uncertainty/mask_per', torch.sum(mask)/mask.numel(), iter_num)
        writer.add_scalar('uncertainty/threshold', threshold, iter_num)
        writer.add_scalar('lr', lr_, iter_num)
        writer.add_scalar('loss/loss', loss, iter_num)
        writer.add_scalar('loss/loss_seg', loss_seg, iter_num)
        writer.add_scalar('loss/loss_seg_dice', loss_seg_dice, iter_num)
        writer.add_scalar('train/consistency_loss', consistency_loss, iter_num)
        writer.add_scalar('train/consistency_weight', consistency_weight, iter_num)
        writer.add_scalar('train/consistency_dist', consistency_dist, iter_num)

        logging.info('iteration %d : loss : %f cons_dist: %f, loss_weight: %f' %
                     (iter_num, loss.item(), consistency_dist.item(), consistency_weight))
        if iter_num % 50 == 0:
            image = volume_batch[0, 0:1, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            grid_image = make_grid(image, 5, normalize=True)
            writer.add_image('train/Image', grid_image, iter_num)

            # image = outputs_soft[0, 3:4, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            image = torch.max(outputs_soft[0, :, :, :, 20:61:10], 0)[1].permute(2, 0, 1).data.cpu().numpy()
```

```
            image = utils.decode_seg_map_sequence(image)
            grid_image = make_grid(image, 5, normalize=False)
            writer.add_image('train/Predicted_label', grid_image, iter_num)

            image = label_batch[0, :, :, 20:61:10].permute(2, 0, 1)
            grid_image = make_grid(utils.decode_seg_map_sequence(image.data.cpu().numpy()), 5, normalize=False)
            writer.add_image('train/Groundtruth_label', grid_image, iter_num)

            image = uncertainty[0, 0:1, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            grid_image = make_grid(image, 5, normalize=True)
            writer.add_image('train/uncertainty', grid_image, iter_num)

            mask2 = (uncertainty > threshold).float()
            image = mask2[0, 0:1, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            grid_image = make_grid(image, 5, normalize=True)
            writer.add_image('train/mask', grid_image, iter_num)
            #####
            image = volume_batch[-1, 0:1, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            grid_image = make_grid(image, 5, normalize=True)
            writer.add_image('unlabel/Image', grid_image, iter_num)

            # image = outputs_soft[-1, 3:4, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            image = torch.max(outputs_soft[-1, :, :, :, 20:61:10], 0)[1].permute(2, 0, 1).data.cpu().numpy()
            image = utils.decode_seg_map_sequence(image)
            grid_image = make_grid(image, 5, normalize=False)
            writer.add_image('unlabel/Predicted_label', grid_image, iter_num)

            image = label_batch[-1, :, :, 20:61:10].permute(2, 0, 1)
            grid_image = make_grid(utils.decode_seg_map_sequence(image.data.cpu().numpy()), 5, normalize=False)
            writer.add_image('unlabel/Groundtruth_label', grid_image, iter_num)
```

```
            ## change lr
            if iter_num % 2500 == 0:
                lr_ = base_lr * 0.1 ** (iter_num // 2500)
                for param_group in optimizer.param_groups:
                    param_group['lr'] = lr_
            if iter_num % 1000 == 0:
                save_mode_path = os.path.join(snapshot_path, 'iter_' + str(iter_num) + '.pth')
                torch.save(model.state_dict(), save_mode_path)
                logging.info("save model to {}".format(save_mode_path))

            if iter_num >= max_iterations:
                break
            time1 = time.time()
        if iter_num >= max_iterations:
            break
    save_mode_path = os.path.join(snapshot_path, 'iter_'+str(max_iterations)+'.pth')
    torch.save(model.state_dict(), save_mode_path)
    logging.info("save model to {}".format(save_mode_path))
    writer.close()
import os
import sys
from tqdm import tqdm
from tensorboardX import SummaryWriter
import shutil
import argparse
import logging
import time
import random
import numpy as np

import torch
```

```
import torch.optim as optim
from torchvision import transforms
import torch.nn.functional as F
import torch.backends.cudnn as cudnn
from torch.utils.data import DataLoader
from torchvision.utils import make_grid

from networks.vnet import VNet
from dataloaders import utils
from utils import ramps, losses
from dataloaders.la_heart import LAHeart, RandomCrop, CenterCrop, RandomRotFlip, ToTensor, TwoStreamBatchSampler



parser = argparse.ArgumentParser()
parser.add_argument('--root_path', type=str, default='../data/cardiography/', help='Name of Experiment')
parser.add_argument('--exp', type=str,  default='UAMT_unlabel', help='model_name')
parser.add_argument('--max_iterations', type=int,  default=6000, help='maximum epoch number to train')
parser.add_argument('--batch_size', type=int, default=4, help='batch_size per gpu')
parser.add_argument('--labeled_bs', type=int, default=2, help='labeled_batch_size per gpu')
parser.add_argument('--base_lr',  type=float,  default=0.01, help='maximum epoch number to train')
parser.add_argument('--deterministic',  type=int,  default=1, help='whether use deterministic training')
parser.add_argument('--seed', type=int,  default=1337, help='random seed')
parser.add_argument('--gpu', type=str,  default='0', help='GPU to use')
### costs
parser.add_argument('--ema_decay', type=float,  default=0.99, help='ema_decay')
parser.add_argument('--consistency_type', type=str,  default="mse", help='consistency_type')
parser.add_argument('--consistency', type=float,  default=0.1, help='consistency')
parser.add_argument('--consistency_rampup', type=float, default=40.0, help='consistency_rampup')
```

```
args = parser.parse_args()

train_data_path = args.root_path
snapshot_path = "../model/" + args.exp + "/"


os.environ['CUDA_VISIBLE_DEVICES'] = args.gpu
batch_size = args.batch_size * len(args.gpu.split(','))
max_iterations = args.max_iterations
base_lr = args.base_lr
labeled_bs = args.labeled_bs

if args.deterministic:
    cudnn.benchmark = False
    cudnn.deterministic = True
    random.seed(args.seed)
    np.random.seed(args.seed)
    torch.manual_seed(args.seed)
    torch.cuda.manual_seed(args.seed)

num_classes = 2
patch_size = (112, 112, 80)


def get_current_consistency_weight(epoch):
    # Consistency ramp-up from https://arxiv.org/abs/1610.02242
    return args.consistency * ramps.sigmoid_rampup(epoch, args.consistency_rampup)

def update_ema_variables(model, ema_model, alpha, global_step):
    # Use the true average until the exponential average is more correct
    alpha = min(1 - 1 / (global_step + 1), alpha)
    for ema_param, param in zip(ema_model.parameters(), model.parameters()):
```

```
        ema_param.data.mul_(alpha).add_(1 - alpha, param.data)


if __name__ == "__main__":
    ## make logger file
    if not os.path.exists(snapshot_path):
        os.makedirs(snapshot_path)
    if os.path.exists(snapshot_path + '/code'):
        shutil.rmtree(snapshot_path + '/code')
    shutil.copytree('.', snapshot_path + '/code', shutil.ignore_patterns(['.git','__pycache__']))

    logging.basicConfig(filename=snapshot_path+"/log.txt", level=logging.INFO,
                        format='[%(asctime)s.%(msecs)03d] %(message)s', datefmt='%H:%M:%S')
    logging.getLogger().addHandler(logging.StreamHandler(sys.stdout))
    logging.info(str(args))

    def create_model(ema=False):
        # Network definition
        net = VNet(n_channels=1, n_classes=num_classes, normalization='batchnorm',
has_dropout=True)
        model = net.cuda()
        if ema:
            for param in model.parameters():
                param.detach_()
        return model

    model = create_model()
    ema_model = create_model(ema=True)

    db_train = LAHeart(base_dir=train_data_path,
                       split='train',
                       transform = transforms.Compose([
                         RandomRotFlip(),
```

```
                    RandomCrop(patch_size),
                    ToTensor(),
                    ]))
    db_test = LAHeart(base_dir=train_data_path,
                      split='test',
                      transform = transforms.Compose([
                        CenterCrop(patch_size),
                        ToTensor()
                      ]))
    labeled_idxs = list(range(16))
    unlabeled_idxs = list(range(16, 80))
    batch_sampler = TwoStreamBatchSampler(labeled_idxs, unlabeled_idxs, batch_size, batch_size-labeled_bs)
    def worker_init_fn(worker_id):
        random.seed(args.seed+worker_id)
    trainloader = DataLoader(db_train, batch_sampler=batch_sampler, num_workers=4, pin_memory=True, worker_init_fn=worker_init_fn)


    model.train()
    ema_model.train()
    optimizer = optim.SGD(model.parameters(), lr=base_lr, momentum=0.9, weight_decay=0.0001)


    if args.consistency_type == 'mse':
        consistency_criterion = losses.softmax_mse_loss
    elif args.consistency_type == 'kl':
        consistency_criterion = losses.softmax_kl_loss
    else:
        assert False, args.consistency_type


    writer = SummaryWriter(snapshot_path+'/log')
    logging.info("{} itertations per epoch".format(len(trainloader)))
```

```
    iter_num = 0
    max_epoch = max_iterations//len(trainloader)+1
    lr_ = base_lr
    model.train()
    for epoch_num in tqdm(range(max_epoch), ncols=70):
        time1 = time.time()
        for i_batch, sampled_batch in enumerate(trainloader):
            time2 = time.time()
            # print('fetch data cost {}'.format(time2-time1))
            volume_batch, label_batch = sampled_batch['image'], sampled_batch['label']
            volume_batch, label_batch = volume_batch.cuda(), label_batch.cuda()
            unlabeled_volume_batch = volume_batch[labeled_bs:]


            noise = torch.clamp(torch.randn_like(unlabeled_volume_batch) * 0.1, -0.2, 0.2)
            ema_inputs = unlabeled_volume_batch + noise
            outputs = model(volume_batch)
            with torch.no_grad():
                ema_output = ema_model(ema_inputs)
            T = 8
            volume_batch_r = unlabeled_volume_batch.repeat(2, 1, 1, 1, 1)
            stride = volume_batch_r.shape[0] // 2
            preds = torch.zeros([stride * T, 2, 112, 112, 80]).cuda()
            for i in range(T//2):
                ema_inputs = volume_batch_r + torch.clamp(torch.randn_like(volume_batch_r) * 0.1, -
0.2, 0.2)
                with torch.no_grad():
                    preds[2 * stride * i:2 * stride * (i + 1)] = ema_model(ema_inputs)
            preds = F.softmax(preds, dim=1)
            preds = preds.reshape(T, stride, 2, 112, 112, 80)
            preds = torch.mean(preds, dim=0)  #(batch, 2, 112,112,80)
            uncertainty = -1.0*torch.sum(preds*torch.log(preds + 1e-6), dim=1, keepdim=True)
#(batch, 1, 112,112,80)
```

```
        ## calculate the loss
        loss_seg = F.cross_entropy(outputs[:labeled_bs], label_batch[:labeled_bs])
        outputs_soft = F.softmax(outputs, dim=1)
        loss_seg_dice = losses.dice_loss(outputs_soft[:labeled_bs, 1, :, :, :], label_batch[:labeled_bs] == 1)
        supervised_loss = 0.5*(loss_seg+loss_seg_dice)

        consistency_weight = get_current_consistency_weight(iter_num//150)
        consistency_dist = consistency_criterion(outputs[labeled_bs:], ema_output) #(batch, 2, 112,112,80)
        threshold = (0.75+0.25*ramps.sigmoid_rampup(iter_num, max_iterations))*np.log(2)
        mask = (uncertainty<threshold).float()
        consistency_dist = torch.sum(mask*consistency_dist)/(2*torch.sum(mask)+1e-16)
        consistency_loss = consistency_weight * consistency_dist
        loss = supervised_loss + consistency_loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        update_ema_variables(model, ema_model, args.ema_decay, iter_num)

        iter_num = iter_num + 1
        writer.add_scalar('uncertainty/mean', uncertainty[0,0].mean(), iter_num)
        writer.add_scalar('uncertainty/max', uncertainty[0,0].max(), iter_num)
        writer.add_scalar('uncertainty/min', uncertainty[0,0].min(), iter_num)
        writer.add_scalar('uncertainty/mask_per', torch.sum(mask)/mask.numel(), iter_num)
        writer.add_scalar('uncertainty/threshold', threshold, iter_num)
        writer.add_scalar('lr', lr_, iter_num)
        writer.add_scalar('loss/loss', loss, iter_num)
        writer.add_scalar('loss/loss_seg', loss_seg, iter_num)
```

```
        writer.add_scalar('loss/loss_seg_dice', loss_seg_dice, iter_num)
        writer.add_scalar('train/consistency_loss', consistency_loss, iter_num)
        writer.add_scalar('train/consistency_weight', consistency_weight, iter_num)
        writer.add_scalar('train/consistency_dist', consistency_dist, iter_num)

        logging.info('iteration %d : loss : %f cons_dist: %f, loss_weight: %f' %
                     (iter_num, loss.item(), consistency_dist.item(), consistency_weight))
        if iter_num % 50 == 0:
            image = volume_batch[0, 0:1, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            grid_image = make_grid(image, 5, normalize=True)
            writer.add_image('train/Image', grid_image, iter_num)

            # image = outputs_soft[0, 3:4, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            image = torch.max(outputs_soft[0, :, :, :, 20:61:10], 0)[1].permute(2, 0, 1).data.cpu().numpy()
            image = utils.decode_seg_map_sequence(image)
            grid_image = make_grid(image, 5, normalize=False)
            writer.add_image('train/Predicted_label', grid_image, iter_num)

            image = label_batch[0, :, :, 20:61:10].permute(2, 0, 1)
            grid_image = make_grid(utils.decode_seg_map_sequence(image.data.cpu().numpy()), 5, normalize=False)
            writer.add_image('train/Groundtruth_label', grid_image, iter_num)

            image = uncertainty[0, 0:1, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            grid_image = make_grid(image, 5, normalize=True)
            writer.add_image('train/uncertainty', grid_image, iter_num)

            mask2 = (uncertainty > threshold).float()
            image = mask2[0, 0:1, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            grid_image = make_grid(image, 5, normalize=True)
            writer.add_image('train/mask', grid_image, iter_num)
```

```
            #####
            image = volume_batch[-1, 0:1, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            grid_image = make_grid(image, 5, normalize=True)
            writer.add_image('unlabel/Image', grid_image, iter_num)

            # image = outputs_soft[-1, 3:4, :, :, 20:61:10].permute(3, 0, 1, 2).repeat(1, 3, 1, 1)
            image = torch.max(outputs_soft[-1, :, :, :, 20:61:10], 0)[1].permute(2, 0, 1).data.cpu().numpy()
            image = utils.decode_seg_map_sequence(image)
            grid_image = make_grid(image, 5, normalize=False)
            writer.add_image('unlabel/Predicted_label', grid_image, iter_num)

            image = label_batch[-1, :, :, 20:61:10].permute(2, 0, 1)
            grid_image = make_grid(utils.decode_seg_map_sequence(image.data.cpu().numpy()), 5, normalize=False)
            writer.add_image('unlabel/Groundtruth_label', grid_image, iter_num)

        ## change lr
        if iter_num % 2500 == 0:
            lr_ = base_lr * 0.1 ** (iter_num // 2500)
            for param_group in optimizer.param_groups:
                param_group['lr'] = lr_
        if iter_num % 1000 == 0:
            save_mode_path = os.path.join(snapshot_path, 'iter_' + str(iter_num) + '.pth')
            torch.save(model.state_dict(), save_mode_path)
            logging.info("save model to {}".format(save_mode_path))

        if iter_num >= max_iterations:
            break
        time1 = time.time()
    if iter_num >= max_iterations:
        break
```

```
save_mode_path = os.path.join(snapshot_path, 'iter_'+str(max_iterations)+'.pth')
torch.save(model.state_dict(), save_mode_path)
logging.info("save model to {}".format(save_mode_path))
writer.close()
```