DL4H_Team_104

April 14, 2024

1 Before you use this template

This template is just a recommended template for project Report. It only considers the general type of research in our paper pool. Feel free to edit it to better fit your project. You will iteratively update the same notebook submission for your draft and the final submission. Please check the project rubriks to get a sense of what is expected in the template.

---

2 FAQ and Attentions

- Copy and move this template to your Google Drive. Name your notebook by your team ID (upper-left corner). Don't eidt this original file.
- This template covers most questions we want to ask about your reproduction experiment. You don't need to exactly follow the template, however, you should address the questions. Please feel free to customize your report accordingly.
- any report must have run-able codes and necessary annotations (in text and code comments).
- The notebook is like a demo and only uses small-size data (a subset of original data or processed data), the entire runtime of the notebook including data reading, data process, model training, printing, figure plotting, etc, must be within 8 min, otherwise, you may get penalty on the grade.
  - If the raw dataset is too large to be loaded you can select a subset of data and pre-process the data, then, upload the subset or processed data to Google Drive and load them in this notebook.
  - If the whole training is too long to run, you can only set the number of training epoch to a small number, e.g., 3, just show that the training is runable.
  - For results model validation, you can train the model outside this notebook in advance, then, load pretrained model and use it for validation (display the figures, print the metrics).
- The post-process is important! For post-process of the results,please use plots/figures. The code to summarize results and plot figures may be tedious, however, it won't be waste of time since these figures can be used for presentation. While plotting in code, the figures should have titles or captions if necessary (e.g., title your figure with "Figure 1. xxxx")
- There is not page limit to your notebook report, you can also use separate notebooks for the report, just make sure your grader can access and run/test them.
- If you use outside resources, please refer them (in any formats). Include the links to the resources if necessary.

3 Link to Original Git Repository

https://github.com/pranavsinghps1/CASS

4 Link to Our Git Repository

https://github.com/TonyDeng1997/CASS_UIUC598DLH

It contains the checkpoints to be used with our model as well as all the required dependencies

5 Environment setup

[1]:
```
!pip3 install -r requirements.txt
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: einops==0.4.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 1)) (0.4.1)
Requirement already satisfied: matplotlib==3.5.2 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 2)) (3.5.2)
Requirement already satisfied: matplotlib-inline==0.1.2 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 3)) (0.1.2)
Requirement already satisfied: numpy==1.23.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 4)) (1.23.1)
Requirement already satisfied: pandas==1.4.3 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 5)) (1.4.3)
Requirement already satisfied: Pillow==9.2.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 6)) (9.2.0)
Requirement already satisfied: scikit-learn==1.1.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 7)) (1.1.1)
Requirement already satisfied: scipy==1.8.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 8)) (1.8.1)
Requirement already satisfied: tensorboard==2.9.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 9)) (2.9.1)
Requirement already satisfied: timm==0.5.4 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 10)) (0.5.4)
Requirement already satisfied: torch==1.11.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 11)) (1.11.0)
Requirement already satisfied: torchaudio==0.11.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 12)) (0.11.0)
Requirement already satisfied: torchcontrib==0.0.2 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 13)) (0.0.2)
Requirement already satisfied: torchmetrics==0.9.2 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 14)) (0.9.2)
Requirement already satisfied: torchvision==0.12.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 15)) (0.12.0)
Requirement already satisfied: vit-pytorch==0.35.8 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 16)) (0.35.8)
Requirement already satisfied: pytorch-lightning==1.6.5 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 17)) (1.6.5)
Requirement already satisfied: tqdm==4.64.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from -r requirements.txt
(line 18)) (4.64.0)
Requirement already satisfied: packaging>=20.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from matplotlib==3.5.2->-r
requirements.txt (line 2)) (24.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from matplotlib==3.5.2->-r
requirements.txt (line 2)) (1.4.5)
Requirement already satisfied: cycler>=0.10 in
/home/hdeng11/.local/lib/python3.10/site-packages (from matplotlib==3.5.2->-r
requirements.txt (line 2)) (0.12.1)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/lib/python3/dist-
packages (from matplotlib==3.5.2->-r requirements.txt (line 2)) (2.4.7)
Requirement already satisfied: fonttools>=4.22.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from matplotlib==3.5.2->-r
requirements.txt (line 2)) (4.51.0)
Requirement already satisfied: python-dateutil>=2.7 in
/home/hdeng11/.local/lib/python3.10/site-packages (from matplotlib==3.5.2->-r
requirements.txt (line 2)) (2.9.0.post0)
Requirement already satisfied: traitlets in
/home/hdeng11/.local/lib/python3.10/site-packages (from matplotlib-
inline==0.1.2->-r requirements.txt (line 3)) (5.14.2)
Requirement already satisfied: pytz>=2020.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from pandas==1.4.3->-r
requirements.txt (line 5)) (2024.1)
Requirement already satisfied: joblib>=1.0.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-learn==1.1.1->-r
requirements.txt (line 7)) (1.4.0)
```

```
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-learn==1.1.1->-r
requirements.txt (line 7)) (3.4.0)
Requirement already satisfied: markdown>=2.6.8 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (2.31.0)
Requirement already satisfied: werkzeug>=1.0.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (3.0.2)
Requirement already satisfied: grpcio>=1.24.3 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (1.62.1)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (1.8.1)
Requirement already satisfied: absl-py>=0.4 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (2.1.0)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (3.19.6)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (0.4.6)
Requirement already satisfied: setuptools>=41.0.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (69.2.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (2.29.0)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from tensorboard==2.9.1->-r
requirements.txt (line 9)) (0.6.1)
Requirement already satisfied: wheel>=0.26 in /usr/lib/python3/dist-packages
(from tensorboard==2.9.1->-r requirements.txt (line 9)) (0.37.1)
Requirement already satisfied: typing-extensions in
/home/hdeng11/.local/lib/python3.10/site-packages (from torch==1.11.0->-r
requirements.txt (line 11)) (4.11.0)
Requirement already satisfied: fsspec[http]!=2021.06.0,>=2021.05.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from pytorch-
lightning==1.6.5->-r requirements.txt (line 17)) (2024.3.1)
Requirement already satisfied: PyYAML>=5.4 in /usr/lib/python3/dist-packages
(from pytorch-lightning==1.6.5->-r requirements.txt (line 17)) (5.4.1)
Requirement already satisfied: pyDeprecate>=0.3.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from pytorch-
lightning==1.6.5->-r requirements.txt (line 17)) (0.3.2)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
fsspec[http]!=2021.06.0,>=2021.05.0->pytorch-lightning==1.6.5->-r
requirements.txt (line 17)) (3.9.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from google-
auth<3,>=1.6.3->tensorboard==2.9.1->-r requirements.txt (line 9)) (0.4.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from google-
auth<3,>=1.6.3->tensorboard==2.9.1->-r requirements.txt (line 9)) (5.3.3)
Requirement already satisfied: rsa<5,>=3.1.4 in
/home/hdeng11/.local/lib/python3.10/site-packages (from google-
auth<3,>=1.6.3->tensorboard==2.9.1->-r requirements.txt (line 9)) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tensorboard==2.9.1->-r requirements.txt (line 9)) (2.0.0)
Requirement already satisfied: six>=1.5 in
/home/hdeng11/.local/lib/python3.10/site-packages (from python-
dateutil>=2.7->matplotlib==3.5.2->-r requirements.txt (line 2)) (1.12.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
requests<3,>=2.21.0->tensorboard==2.9.1->-r requirements.txt (line 9)) (3.3.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
requests<3,>=2.21.0->tensorboard==2.9.1->-r requirements.txt (line 9)) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
requests<3,>=2.21.0->tensorboard==2.9.1->-r requirements.txt (line 9))
(2024.2.2)
Requirement already satisfied: idna<4,>=2.5 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
requests<3,>=2.21.0->tensorboard==2.9.1->-r requirements.txt (line 9)) (2.8)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
werkzeug>=1.0.1->tensorboard==2.9.1->-r requirements.txt (line 9)) (2.1.5)
Requirement already satisfied: frozenlist>=1.1.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]!=2021.06.0,>=2021.05.0->pytorch-
lightning==1.6.5->-r requirements.txt (line 17)) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]!=2021.06.0,>=2021.05.0->pytorch-
lightning==1.6.5->-r requirements.txt (line 17)) (6.0.5)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]!=2021.06.0,>=2021.05.0->pytorch-
lightning==1.6.5->-r requirements.txt (line 17)) (4.0.3)
```

```
Requirement already satisfied: yarl<2.0,>=1.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]!=2021.06.0,>=2021.05.0->pytorch-
lightning==1.6.5->-r requirements.txt (line 17)) (1.9.4)
Requirement already satisfied: aiosignal>=1.1.2 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]!=2021.06.0,>=2021.05.0->pytorch-
lightning==1.6.5->-r requirements.txt (line 17)) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]!=2021.06.0,>=2021.05.0->pytorch-
lightning==1.6.5->-r requirements.txt (line 17)) (23.2.0)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard==2.9.1->-r
requirements.txt (line 9)) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/lib/python3/dist-packages
(from requests-oauthlib>=0.7.0->google-auth-
oauthlib<0.5,>=0.4.1->tensorboard==2.9.1->-r requirements.txt (line 9)) (3.2.0)
```

[2]:

```
!pip3 install --upgrade git+https://github.com/MedMNIST/MedMNIST.git
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting git+https://github.com/MedMNIST/MedMNIST.git
  Cloning https://github.com/MedMNIST/MedMNIST.git to /tmp/pip-req-
build-0clm6k16
  Running command git clone --filter=blob:none --quiet
https://github.com/MedMNIST/MedMNIST.git /tmp/pip-req-build-0clm6k16
  Resolved https://github.com/MedMNIST/MedMNIST.git to commit
db5bff9d0faef4d273896e3cb5542a7000c0239f
  Preparing metadata (setup.py) … done
Requirement already satisfied: numpy in
/home/hdeng11/.local/lib/python3.10/site-packages (from medmnist==3.0.1)
(1.23.1)
Requirement already satisfied: pandas in
/home/hdeng11/.local/lib/python3.10/site-packages (from medmnist==3.0.1) (1.4.3)
Requirement already satisfied: scikit-learn in
/home/hdeng11/.local/lib/python3.10/site-packages (from medmnist==3.0.1) (1.1.1)
Requirement already satisfied: scikit-image in
/home/hdeng11/.local/lib/python3.10/site-packages (from medmnist==3.0.1)
(0.22.0)
Requirement already satisfied: tqdm in /home/hdeng11/.local/lib/python3.10/site-
packages (from medmnist==3.0.1) (4.64.0)
Requirement already satisfied: Pillow in
/home/hdeng11/.local/lib/python3.10/site-packages (from medmnist==3.0.1) (9.2.0)
Requirement already satisfied: fire in /home/hdeng11/.local/lib/python3.10/site-
packages (from medmnist==3.0.1) (0.6.0)
Requirement already satisfied: torch in
```

```
/home/hdeng11/.local/lib/python3.10/site-packages (from medmnist==3.0.1)
(1.11.0)
Requirement already satisfied: torchvision in
/home/hdeng11/.local/lib/python3.10/site-packages (from medmnist==3.0.1)
(0.12.0)
Requirement already satisfied: termcolor in
/home/hdeng11/.local/lib/python3.10/site-packages (from fire->medmnist==3.0.1)
(2.4.0)
Requirement already satisfied: six in /home/hdeng11/.local/lib/python3.10/site-
packages (from fire->medmnist==3.0.1) (1.12.0)
Requirement already satisfied: pytz>=2020.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from pandas->medmnist==3.0.1)
(2024.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from pandas->medmnist==3.0.1)
(2.9.0.post0)
Requirement already satisfied: scipy>=1.8 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-
image->medmnist==3.0.1) (1.8.1)
Requirement already satisfied: tifffile>=2022.8.12 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-
image->medmnist==3.0.1) (2024.2.12)
Requirement already satisfied: packaging>=21 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-
image->medmnist==3.0.1) (24.0)
Requirement already satisfied: imageio>=2.27 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-
image->medmnist==3.0.1) (2.34.0)
Requirement already satisfied: lazy_loader>=0.3 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-
image->medmnist==3.0.1) (0.4)
Requirement already satisfied: networkx>=2.8 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-
image->medmnist==3.0.1) (3.3)
Requirement already satisfied: joblib>=1.0.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-
learn->medmnist==3.0.1) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/hdeng11/.local/lib/python3.10/site-packages (from scikit-
learn->medmnist==3.0.1) (3.4.0)
Requirement already satisfied: typing-extensions in
/home/hdeng11/.local/lib/python3.10/site-packages (from torch->medmnist==3.0.1)
(4.11.0)
Requirement already satisfied: requests in
/home/hdeng11/.local/lib/python3.10/site-packages (from
torchvision->medmnist==3.0.1) (2.31.0)
Requirement already satisfied: certifi>=2017.4.17 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
requests->torchvision->medmnist==3.0.1) (2024.2.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
requests->torchvision->medmnist==3.0.1) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
requests->torchvision->medmnist==3.0.1) (2.8)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/home/hdeng11/.local/lib/python3.10/site-packages (from
requests->torchvision->medmnist==3.0.1) (1.24.3)
```

6 Introduction

- **Background of the problem:**

Medical image analysis is vitally important in diagnosing diseases and predicting patient outcomes in general. However, the effectiveness of deep learning techniques in this realm is often hindered by a lack of labeled data due to issues such as a need for the domain-specific knowledge required for labeling, patient privacy concerns, disease prevalence, and an incomplete understanding of rare or emerging diseases. This is a challenge when applying deep learning techniques whose performance often relies on large annotated datasets. Additionally, existing state-of-the-art self-supervised learning techniques often require significant computational resources, such as large batch sizes and extensive training times, to achieve maximum performance. This oftentimes makes them impractical for many medical imaging applications where compute budgets are limited.

Current state-of-the-art self-supervised models such as DINO and BYOL can perform well with limited labeled data, but they still require considerable computational resources. DINO is a teacher and student network, where both the student and the teacher utilize a Vision Transformer (ViT) architecture. This state-of-the-art, self-supervised model performed comparatively well to its predecessors when trained on unlabeled data. However, its performance is sometimes dwarfed by supervised models. Developing a deep learning architecture which performs well with limited labeled data and limited computational resources, could lead to increased patient longevity and general health.

- **Paper explanation:**

The key idea proposed in the CASS paper [1] is to leverage both CNNs and Transformers simultaneously, in a self-supervised learning approach, in order to address the challenges of limited labeled data and high computational cost in medical image analysis. Unlike existing self-supervised methods, which may use one of these architectures, CASS passes images through a CNN and Transformer in parallel to extract their representations. These representations are then used to find cosine similarity loss. The idea is that CNNs and Transformers have different strong suits and by training them in parallel they're able to learn from one another. CNNs are translation equivariant and better at capturing local details, while Transformers are better at modeling global context. By contrasting the features extracted by each architecture, they are able to learn from one another and eventually capture a richer and more useful representation.

The authors demonstrate that CASS outperforms existing self-supervised methods, like DINO, in terms of F1 Score and Recall value by an average of 3.8% with 1% labeled data, 5.9% with 10% labeled data, and 10.13% with 100% labeled data [1]. It achieved this while requiring 69% less training time on average, compared to DINO. CASS was also shown to be more robust to changes in batch size and training epochs, which is a key limitation in compute restricted environments.

CASS marks an important step towards making self-supervised deep learning models more widely accessible and practically useful for medical image analysis. It has the potential to accelerate research on rare and emerging diseases where labeled data is scarce, thus advancing the field of computer-aided diagnosis and patient outcome prediction.

7 Scope of Reproducibility:

In our project, we aim to validate the following hypotheses:

- **Hypothesis 1:** CASS requires less training time than existing self-supervised models (DINO).
- **Hypothesis 2:** CASS is more robust to changes in batch size and number of epochs.

To test these hypotheses, we will conduct the following experiments:

1. **Training time comparison:** We will measure and compare the time required to train CASS and DINO on the same dataset. We will be comparing our results to the DINO results presented in the paper Unfortunately, our hardware will not exactly match what was used in the paper, so we will need to make a calculated estimation on what the expected difference would be given the differences between our hardware.
2. **Batch size ablation:** We will evaluate the performance change of CASS when pre-trained with different batch sizes on the same dataset. We hope to test with batch sizes of 8, 16, and 32, though we may be limited by our hardware to do less. We will measure the classification performance and compare results between the varying batch sizes. The paper found that CASS actually performed better with smaller batch sizes and it will be interesting to see if this is reproduced in our own experiments.
3. **Pre-training epochs ablation:** We will measure the effect of the number of pre-training epochs on the performance of CASS when trained on identical datasets. Our goal is to pre-train with epochs of 50, 100, and 200. As in experiment 2, we will measure the classification performance and compare results between the varying epoch ranges. The authors found that there were diminishing returns on performance when increasing the epochs beyond 200, so we will focus our experiments on the lower epoch ranges.

We have less computational resources than the authors did and our experiments may be affected. Because of this, we plan to be flexible in our work and will update our experiments accordingly.

8 Methodology

This methodology is the core of your project. It consists of run-able codes with necessary annotations to show the expeiment you executed for testing the hypotheses.

The methodology at least contains two subsections **data** and **model** in your experiment.

The following section imports the required packages to run the code

[1]:

```
# import  packages you need
import numpy as np
from tqdm import tqdm
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data as data
import torchvision.transforms as transforms
import medmnist
from medmnist import INFO, Evaluator
print(f"MedMNIST v{medmnist.__version__} @ {medmnist.HOMEPAGE}")

print(torch.__version__)
```

```
/home/hdeng11/.local/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.17.3 and <1.25.0 is required for this version
of SciPy (detected version 1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
MedMNIST v3.0.1 @ https://github.com/MedMNIST/MedMNIST/
1.11.0+cu102
```

[75]:

```
import os
import numpy as np
import pytorch_lightning as pl
import torch
import pandas as pd
import timm
import torch.nn as nn
from tqdm import tqdm
from PIL import Image
from sklearn.model_selection import KFold
from torchvision import transforms as tsfm
from torch.utils.data import Dataset, DataLoader
from pytorch_lightning import Trainer, seed_everything
from pytorch_lightning.loggers import CSVLogger
from pytorch_lightning.callbacks import ModelCheckpoint, EarlyStopping
from torchcontrib.optim import SWA
from torchmetrics import Metric
from torch.utils.tensorboard import SummaryWriter
```

[76]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image
from glob import glob
```

```
#from skimage.io import imread
from os import listdir
import time
import copy
from tqdm import tqdm
```

[77]:

```
# General Imports
import matplotlib.pyplot as plt
```

8.1 Data

- The dataset we used is https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset as referred as brain tumor mri dataset in the paper
- The detailed statistics are in the following section, it provided traning and testing split for us
- The dataset didn't provide a label csv file, we'll generate label csv file for both training and testing datasets in this section as well

[78]:

```
training_base_path = "BrainMRI/Training/"
```

[79]:

```
testing_base_path = "BrainMRI/Testing/"
```

[82]:

```
training_folders = listdir(training_base_path)
print(len(training_folders))
```

```
4
```

[86]:

```
training_folders
```

[86]:

```
['glioma', 'meningioma', 'pituitary', 'notumor']
```

[90]:

```
training_file_count=[]
for i in training_folders:
    path = training_base_path + i
    sub_files = listdir(path)
    training_file_count.append(len(sub_files))
```

[91]:

```
testing_folders = listdir(testing_base_path)
print(len(testing_folders))
```

```
4
```

[94]:

```
testing_folders
```

[94]:

```
['glioma', 'meningioma', 'pituitary', 'notumor']
```

[96]:

```
testing_file_count=[]
for i in testing_folders:
    path = testing_base_path + i
    sub_files = listdir(path)
    testing_file_count.append(len(sub_files))
    print(len(sub_files))
```

```
300
306
300
405
```

[98]:

```
total_training_images = 0
for n in range(len(training_folders)):
    patient_id = training_folders[n]
    patient_path = training_base_path  + patient_id
    print(patient_path)
    class_path = patient_path + "/"  + "/"
    subfiles = listdir(class_path)
    total_training_images += len(subfiles)
print("Total Number of Training Images:" + str(total_training_images))
```

```
BrainMRI/Training/glioma
BrainMRI/Training/meningioma
BrainMRI/Training/pituitary
BrainMRI/Training/notumor
Total Number of Training Images:5712
```

[99]:

```
total_testing_images = 0
for n in range(len(testing_folders)):
    patient_id = testing_folders[n]
    patient_path = testing_base_path  + patient_id
    print(patient_path)
    class_path = patient_path + "/"  + "/"
    subfiles = listdir(class_path)
    total_testing_images += len(subfiles)
print("Total Number of Testing Images:" + str(total_testing_images))
```

```
BrainMRI/Testing/glioma
BrainMRI/Testing/meningioma
BrainMRI/Testing/pituitary
BrainMRI/Testing/notumor
Total Number of Testing Images:1311
```

[100]:

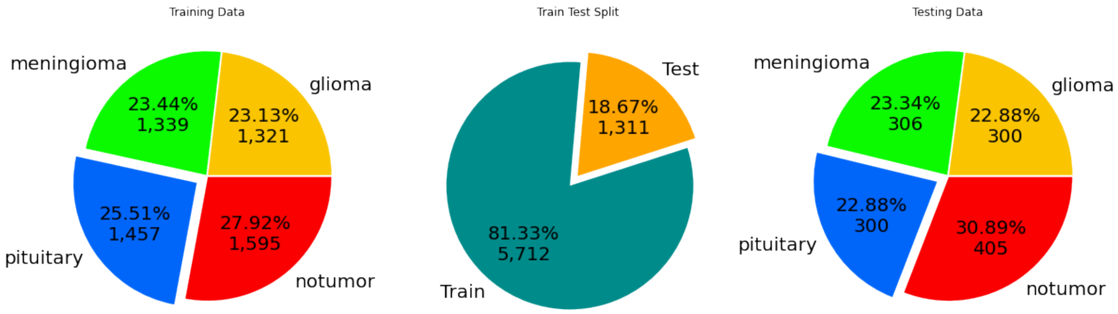
```
testing_file_count
```

[100]:

```
[300, 306, 300, 405]
```

[101]:

```
_, ax = plt.subplots(ncols=3, figsize=(20, 14))

# Plotting training data types

ax[0].set_title('Training Data')
ax[0].pie(
    training_file_count,
    labels=training_folders,
    colors=['#FAC500','#0BFA00', '#0066FA','#FA0000'],
    autopct=lambda p: '{:.2f}%\n{:,.0f}'.format(p, p *total_training_images /␣
 ↪100),
    explode=(0.01, 0.01, 0.1, 0.01),
    textprops={'fontsize': 20}
)



ax[1].set_title('Train Test Split')
ax[1].pie(
    [total_training_images, total_testing_images],
    labels=['Train','Test'],
    colors=['darkcyan', 'orange'],
    autopct=lambda p: '{:.2f}%\n{:,.0f}'.format(p, p *␣
 ↪sum([total_training_images, total_testing_images]) / 100),
    explode=(0.1, 0),
    startangle=85,
    textprops={'fontsize': 20}
)


ax[2].set_title('Testing Data')
ax[2].pie(
    testing_file_count,
    labels=testing_folders,
    colors=['#FAC500', '#0BFA00', '#0066FA', '#FA0000'],
    autopct=lambda p: '{:.2f}%\n{:,.0f}'.format(p, p * total_testing_images /␣
 ↪100),
    explode=(0.01, 0.01, 0.1, 0.01),
    textprops={'fontsize': 20}
)


plt.show()
```



[130]:
```
data = pd.DataFrame(index=np.arange(0, total_training_images), columns=["path",␣
 ↪"target"])

k = 0
for n in range(len(training_folders)):
    patient_id = training_folders[n]
    patient_path = training_base_path  + patient_id
    class_path = patient_path + "/"
    subfiles = listdir(class_path)
    for m in range(len(subfiles)):
        image_path = subfiles[m]
        data.iloc[k]["path"] = class_path + image_path
        data.iloc[k]["target"] = patient_id
        k += 1

data.head()
```

[130]:
```
                                     path  target
0  BrainMRI/Training/glioma/Tr-gl_0162.jpg  glioma
1  BrainMRI/Training/glioma/Tr-gl_0840.jpg  glioma
2  BrainMRI/Training/glioma/Tr-gl_0372.jpg  glioma
3  BrainMRI/Training/glioma/Tr-gl_0343.jpg  glioma
4  BrainMRI/Training/glioma/Tr-gl_0367.jpg  glioma
```

[131]:
```
data['target'].value_counts()/len(data)
```

[131]:
```
notumor       0.279237
pituitary     0.255077
meningioma    0.234419
glioma        0.231268
Name: target, dtype: float64
```

If would like to take a subset of data, do it here

[132]:
```
data_train = data.sample(int(len(data)*1))
```

[133]:

```
data_train['target']
```

[133]:

```
4098     pituitary
871         glioma
3503     pituitary
1765    meningioma
2334    meningioma
           …
1317        glioma
2283    meningioma
2004    meningioma
3668     pituitary
607         glioma
Name: target, Length: 5712, dtype: object
```

[134]:

```
#creating CSV for the entire dataset
data_train.to_csv('BrainMRI/training.csv')
```

Get the class weights to use with focal loss

[143]:

```
label_str2num={}
num=0
for i in data_train['target'].unique():
    label_str2num[i]=num
    num+=1
label_str2num
```

[143]:

```
{'pituitary': 0, 'glioma': 1, 'meningioma': 2, 'notumor': 3}
```

[144]:

```
data={}
for i in data_train['target']:
    if i in data:
        data[i]+=1
    else:
        data[i]=1
```

[145]:

```
data_train['target'].value_counts()
```

[145]:

```
notumor       1595
pituitary     1457
meningioma    1339
glioma        1321
Name: target, dtype: int64
```

[146]:

```
new_data={}
for i in data:
    new_data[label_str2num[i]]=data[i]
```

```
[147]: new_data
```

```
[147]: {0: 1457, 1: 1321, 2: 1339, 3: 1595}
```

```
[148]: from collections import OrderedDict
dist = OrderedDict(sorted(new_data.items()))
dist=dict(dist)
```

```
[149]: def normalize(arr, t_min, t_max):
    norm_arr = []
    diff = t_max - t_min
    diff_arr = max(arr) - min(arr)
    for i in arr:
        temp = (((i - min(arr))*diff)/diff_arr) + t_min
        norm_arr.append(temp)
    return norm_arr

# assign array and range
array_1d = dist.values()
range_to_normalize = (0.2, 1)
normalized_array_1d = normalize(
    array_1d, range_to_normalize[0],
  range_to_normalize[1])

# display original and normalized array
print("Original Array = ", array_1d)
print("Normalized Array = ", normalized_array_1d)
```

```
Original Array =  dict_values([1457, 1321, 1339, 1595])
Normalized Array =  [0.5970802919708029, 0.2, 0.25255474452554744, 1.0]
```

We repeat the same to generate csv file for test datasets

```
[150]: data = pd.DataFrame(index=np.arange(0, total_testing_images), columns=["path", "target"])

k = 0
for n in range(len(testing_folders)):
    patient_id = testing_folders[n]
    patient_path = testing_base_path  + patient_id
    class_path = patient_path + "/"
    subfiles = listdir(class_path)
    for m in range(len(subfiles)):
        image_path = subfiles[m]
        data.iloc[k]["path"] = class_path + image_path
        data.iloc[k]["target"] = patient_id
        k += 1
```

```
data.head()
```

[150]:

```
                                     path  target
0  BrainMRI/Testing/glioma/Te-gl_0262.jpg  glioma
1  BrainMRI/Testing/glioma/Te-gl_0061.jpg  glioma
2  BrainMRI/Testing/glioma/Te-gl_0040.jpg  glioma
3  BrainMRI/Testing/glioma/Te-gl_0287.jpg  glioma
4  BrainMRI/Testing/glioma/Te-gl_0075.jpg  glioma
```

[151]:

```
data['target']
```

[151]:

```
0        glioma
1        glioma
2        glioma
3        glioma
4        glioma
          …
1306    notumor
1307    notumor
1308    notumor
1309    notumor
1310    notumor
Name: target, Length: 1311, dtype: object
```

[152]:

```
data.to_csv('BrainMRI/testing.csv')
```

9 Model

This model configures the CASS pretraining algorithm with furthur fine-tuning with both CNN and Vit. We are reusing the code from here https://github.com/pranavsinghps1/CASS/blob/master/CASS.ipynb and reusing parameters used in paper for draft purpose with minor changes due to different datasets were being used.

9.0.1 If you are interested in the result and would not like to run the training again, please jump to the result section, we've saved a checkpoint to use in the result section

[4]:

```
import os
import numpy as np
import pytorch_lightning as pl
import torch
import pandas as pd
import timm
import math
import torch.nn as nn
from tqdm import tqdm
from PIL import Image
```

```python
from torchvision import transforms as tsfm
from torch.utils.data import Dataset, DataLoader
from pytorch_lightning import Trainer, seed_everything
from pytorch_lightning.loggers import CSVLogger
from pytorch_lightning.callbacks import ModelCheckpoint, EarlyStopping
from torchcontrib.optim import SWA
from torchmetrics import Metric
from torch.utils.tensorboard import SummaryWriter
```

Copy the label_num2str and cls weight to here

[5]:

```python
class CFG:
    # data path
    data_path  = 'BrainMRI/training.csv'
    train_imgs_dir  = 'BrainMRI/Training'
    # model info
    # label info
    label_num2str = {0: 'glioma',
                     1: 'pituitary',
                     2:'notumor',
                     3:'meningioma'
                     }
    label_str2num = {'glioma': 0,
                     'pituitary':1,
                     'notumor':2,
                     'meningioma':3
                     }
    fl_alpha = 1.0  # alpha of focal_loss
    fl_gamma = 2.0  # gamma of focal_loss
    cls_weight =  [0.2, 0.5970802919708029, 1.0, 0.25255474452554744] # copy␣
↪the cls_weight from previous step or just use the variable
    cnn_name='resnet50'
    vit_name='vit_base_patch16_384'
    seed = 77
    num_classes = 4
    batch_size = 16
    t_max = 16
    lr = 1e-3
    min_lr = 1e-6
    n_fold = 6
    num_workers = 8
    gpu_idx = 0
    device = torch.device(f'cuda:{gpu_idx}' if torch.cuda.is_available() else␣
↪'cpu')
    gpu_list = [gpu_idx]
```

[6]:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
seed_everything(77)
cfg=CFG()
```

```
Global seed set to 77
```

Compute Image Mean and Variance to be used for transform function, since we are using the same dataset. We don't really have to run this multiple times.

[20]:

```
from torchvision import datasets, transforms
transform = transforms.Compose([
    transforms.Resize((384, 384)),  # Resize all images to the same size
    transforms.ToTensor()           # Transform images to PyTorch tensors
])
dataset = datasets.ImageFolder('BrainMRI/Training', transform=transform)
```

[21]:

```
from torch.utils.data import DataLoader

dataloader = DataLoader(dataset, batch_size=16, shuffle=False, num_workers=4)
```

[22]:

```
import torch

def get_mean_and_std(dataloader):
    channel_sum, channel_squared_sum, num_elements = 0, 0, 0

    for data, _ in dataloader:
        # Reshape data to be the shape of [B, C, W*H]
        data = data.view(data.size(0), data.size(1), -1)
        # Sum over all pixels and all batches for each channel
        channel_sum += torch.sum(data, dim=[0, 2])
        channel_squared_sum += torch.sum(data ** 2, dim=[0, 2])
        num_elements += data.size(2) * data.size(0)

    # Calculate the mean and variance
    print(num_elements)
    mean = channel_sum / num_elements
    variance = (channel_squared_sum / num_elements) - (mean ** 2)
    std_dev = torch.sqrt(variance)

    return mean, std_dev

# Calculate mean and std
mean, std = get_mean_and_std(dataloader)
print(f'Mean: {mean}')
print(f'Std Deviation: {std}')
```

```
842268672
Mean: tensor([0.1857, 0.1857, 0.1858])
```

```
Std Deviation: tensor([0.2018, 0.2018, 0.2018])
```

[10]:

```
"""
Define train & valid image transformation
"""
# The mean and std from previous step goes here
DATASET_IMAGE_MEAN = (0.1857, 0.1857, 0.1858)
DATASET_IMAGE_STD = (0.2018, 0.2018, 0.2018)

train_transform = tsfm.Compose([tsfm.Resize((384,384)),
                                tsfm.RandomApply([tsfm.ColorJitter(0.2, 0.2, 0.
 ↪2),tsfm.RandomPerspective(distortion_scale=0.2),], p=0.3),
                                tsfm.RandomApply([tsfm.ColorJitter(0.2, 0.2, 0.
 ↪2),tsfm.RandomAffine(degrees=10),], p=0.3),
                                tsfm.RandomVerticalFlip(p=0.3),
                                tsfm.RandomHorizontalFlip(p=0.3),
                                tsfm.ToTensor(),
                                tsfm.Normalize(DATASET_IMAGE_MEAN,␣
 ↪DATASET_IMAGE_STD), ])

valid_transform = tsfm.Compose([tsfm.Resize((384,384)),
                                tsfm.ToTensor(),
                                tsfm.Normalize(DATASET_IMAGE_MEAN,␣
 ↪DATASET_IMAGE_STD), ])
```

[6]:

```
"""
Define dataset class
"""
class Dataset(Dataset):
    def __init__(self, cfg, img_names: list, labels: list, transform=None):
        self.img_dir = cfg.train_imgs_dir
        self.img_names = img_names
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.img_names)

    def __getitem__(self, idx):
        img_path = self.img_names[idx]
        img = Image.open(img_path).convert('RGB')
        img_ts = self.transform(img)
        label_ts = self.labels[idx]
        return img_ts, label_ts
```

[7]:

```
"""
Define Focal-Loss
```

```
"""

class FocalLoss(nn.Module):
    """
    The focal loss for fighting against class-imbalance
    """
    def __init__(self, alpha=1, gamma=2):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = 1e-12  # prevent training from Nan-loss error
        self.cls_weights = torch.tensor([CFG.cls_weight],dtype=torch.float,␣
 ↪requires_grad=False, device=CFG.device)

    def forward(self, logits, target):
        """
        logits & target should be tensors with shape [batch_size, num_classes]
        """
        probs = torch.sigmoid(logits)
        one_subtract_probs = 1.0 - probs
        # add epsilon
        probs_new = probs + self.epsilon
        one_subtract_probs_new = one_subtract_probs + self.epsilon
        # calculate focal loss
        log_pt = target * torch.log(probs_new) + (1.0 - target) * torch.
 ↪log(one_subtract_probs_new)
        pt = torch.exp(log_pt)
        focal_loss = -1.0 * (self.alpha * (1 - pt) ** self.gamma) * log_pt
        focal_loss = focal_loss * self.cls_weights
        return torch.mean(focal_loss)
```

[8]:

```
"""
Define F1 score metric
"""
class MyF1Score(Metric):
    def __init__(self, cfg, threshold: float = 0.5, dist_sync_on_step=False):
        super().__init__(dist_sync_on_step=dist_sync_on_step)
        self.cfg = cfg
        self.threshold = threshold
        self.add_state("tp", default=torch.tensor(0), dist_reduce_fx="sum")
        self.add_state("fp", default=torch.tensor(0), dist_reduce_fx="sum")
        self.add_state("fn", default=torch.tensor(0), dist_reduce_fx="sum")

    def update(self, preds: torch.Tensor, target: torch.Tensor):
        assert preds.shape == target.shape
        preds_str_batch = self.num_to_str(torch.sigmoid(preds))
        target_str_batch = self.num_to_str(target)
        tp, fp, fn = 0, 0, 0
        for pred_str_list, target_str_list in zip(preds_str_batch,␣
 ↪target_str_batch):
            for pred_str in pred_str_list:
                if pred_str in target_str_list:
                    tp += 1
                if pred_str not in target_str_list:
                    fp += 1

            for target_str in target_str_list:
                if target_str not in pred_str_list:
                    fn += 1
        self.tp += tp
        self.fp += fp
        self.fn += fn

    def compute(self):
        #To switch between F1 score and recall.
        #f1 = 2.0 * self.tp / (2.0 * self.tp + self.fn + self.fp)
        rec = self.tp/(self.tp + self.fn)
        return rec

    def num_to_str(self, ts: torch.Tensor) -> list:
        batch_bool_list = (ts > self.threshold).detach().cpu().numpy().tolist()
        batch_str_list = []
        for one_sample_bool in batch_bool_list:
            lb_str_list = [self.cfg.label_num2str[lb_idx] for lb_idx, bool_val␣
 ↪in enumerate(one_sample_bool) if bool_val]
            batch_str_list.append(lb_str_list)
        return batch_str_list
```

[14]:

```
df=pd.read_csv(cfg.data_path)
```

[15]:

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(df['path'], df['target'],␣
 ↪test_size=0.2, random_state=77)
```

[16]:

```
all_img_names: list = X_train.values.tolist()
all_img_names_valid: list = X_val.values.tolist()
```

[17]:

```
len(all_img_names)
```

[17]:

```
4569
```

[18]:

```
len(all_img_names_valid)
```

[18]:

```
1143
```

[19]:

```
len(all_img_names) + len(all_img_names_valid)
```

[19]:

```
5712
```

[20]:

```
all_img_labels_ts = []
for tmp_lb in y_train:
    tmp_label = torch.zeros([CFG.num_classes], dtype=torch.float)
    label_num=CFG.label_str2num.get(tmp_lb)
    k=int(label_num)
    tmp_label[k] = 1.0
    all_img_labels_ts.append(tmp_label)
```

[21]:

```
all_img_labels_val_ts = []
for tmp_lb in y_val:
    tmp_label = torch.zeros([CFG.num_classes], dtype=torch.float)
    label_num=CFG.label_str2num.get(tmp_lb)
    k=int(label_num)
    tmp_label[k] = 1.0
    all_img_labels_val_ts.append(tmp_label)
```

[23]:

```
model_cnn = timm.create_model(cfg.cnn_name, pretrained=True)
model_vit = timm.create_model(cfg.vit_name, pretrained=True)
model_cnn.to(device)
model_vit.to(device)
```

[23]:

```
VisionTransformer(
  (patch_embed): PatchEmbed(
    (proj): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
    (norm): Identity()
  )
  (pos_drop): Dropout(p=0.0, inplace=False)
  (blocks): Sequential(
    (0): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (1): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (2): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (3): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (4): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (5): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (6): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (7): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (8): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (9): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (10): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (11): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
```

```
    )
  )
  (norm): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
  (pre_logits): Identity()
  (head): Linear(in_features=768, out_features=1000, bias=True)
)
```

[24]:

```
def␣
 ↪ssl_train_model(train_loader,model_vit,criterion_vit,optimizer_vit,scheduler_vit,model_cnn,
 ↪
    writer = SummaryWriter()
    phase = 'train'
    model_cnn.train()
    model_vit.train()
    f1_score_cnn=0
    f1_score_vit=0
    for i in tqdm(range(num_epochs)):
        with torch.set_grad_enabled(phase == 'train'):
            for img,_ in train_loader:
                f1_score_cnn=0
                f1_score_vit=0
                img = img.to(device)
                pred_vit = model_vit(img)
                pred_cnn = model_cnn(img)
                model_sim_loss=loss_fn(pred_vit,pred_cnn)
                loss = model_sim_loss.mean()
                loss.backward()
                optimizer_cnn.step()
                optimizer_vit.step()
                scheduler_cnn.step()
                scheduler_vit.step()
            print('For -',i,'Loss:',loss)
            writer.add_scalar("Self-Supervised Loss/train", loss, i)
    writer.flush()
```

[25]:

```
optimizer_cnn = SWA(torch.optim.Adam(model_cnn.parameters(), lr= 1e-3))
optimizer_vit = SWA(torch.optim.Adam(model_vit.parameters(), lr= 1e-3))
scheduler_cnn = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer_cnn,
                                                                     T_max=16,
                                                                    ␣
 ↪eta_min=1e-6)
scheduler_vit = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer_vit,
                                                                     T_max=16,
                                                                    ␣
 ↪eta_min=1e-6)

criterion_vit = FocalLoss(cfg.fl_alpha, cfg.fl_gamma)
```

```
criterion_cnn = FocalLoss(cfg.fl_alpha, cfg.fl_gamma)
```

[26]:

```
def loss_fn(x, y):
    x =  torch.nn.functional.normalize(x, dim=-1, p=2)
    y =  torch.nn.functional.normalize(y, dim=-1, p=2)
    return 2 - 2 * (x * y).sum(dim=-1)
```

[27]:

```
import random
random.seed(77)
x=0.1 #currently set to use 10% of the labels for reduced label training
onep=random.sample(range(0, len(X_train)), int(len(X_train)*x))
all_img_names_train = [all_img_names[idx] for idx in onep]
all_img_labels_ts_train = [all_img_labels_ts[idx] for idx in onep]
```

[28]:

```
train_dataset = Dataset(CFG, all_img_names_train,all_img_labels_ts_train,␣
 ↪train_transform)
valid_dataset = Dataset(CFG, all_img_names_valid, all_img_labels_val_ts,␣
 ↪valid_transform)
train_loader = DataLoader(train_dataset, batch_size=CFG.batch_size,␣
 ↪shuffle=True, num_workers=CFG.num_workers)
valid_loader = DataLoader(valid_dataset, batch_size=CFG.batch_size,␣
 ↪shuffle=False, num_workers=CFG.num_workers)
```

[29]:

```
len(valid_dataset)
```

[29]:

```
1143
```

[30]:

```
len(train_dataset)
```

[30]:

```
456
```

ssl_train_model will return two models (one CNN one Vit) for futurue fine tuning, while DINO only returns one

[ ]:

```
#Train SSL
print('Training Cov-T')
#Change Epoche Here
ssl_train_model(train_loader,model_vit,criterion_vit,optimizer_vit,scheduler_vit,model_cnn,cri
#Saving SSL Models
print('Saving Cov-T')
torch.save(model_cnn,'./cass-r50-isic.pt')
torch.save(model_vit,'./cass-r50-vit-isic.pt')
```

```
Training Cov-T

  0%|
| 0/1 [00:00<?, ?it/s]
```

[32]:

```
model_cnn=torch.load('cass-r50-isic.pt')
model_vit=torch.load('cass-r50-vit-isic.pt')
```

[252]:

```
from torch.autograd import Variable
#Train Correspong Supervised CNN
print('Fine tunning Cov-T')
model_cnn.fc=nn.Linear(in_features=2048, out_features=4, bias=True)
criterion = FocalLoss(cfg.fl_alpha, cfg.fl_gamma)
metric = MyF1Score(cfg)
val_metric=MyF1Score(cfg)
optimizer = torch.optim.Adam(model_cnn.parameters(), lr = 3e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,T_max=cfg.
 ↪t_max,eta_min=cfg.min_lr,verbose=True)
model_cnn.train()
best=0
best_val=0
last_loss=math.inf
writer = SummaryWriter()
#change Epoch Here
for epoch in range(1):
    for images,label in train_loader:
        model_cnn.train()
        images = images.to(device)
        label = label.to(device)
        model_cnn.to(device)
        pred_ts=model_cnn(images)
        print(pred_ts)
        print(label)
        loss = criterion(pred_ts, label)
        score = metric(pred_ts, label)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        scheduler.step()
    train_score=metric.compute()
    logs = {'train_loss': loss, 'Recall': train_score, 'lr': optimizer.
 ↪param_groups[0]['lr']}
    writer.add_scalar("Supervised-CNN Loss/train", loss, epoch)
    writer.add_scalar("Supervised-CNN Recall/train", train_score, epoch)

    print(logs)
    if best < train_score:
        with torch.no_grad():
            best=train_score
            model_cnn.eval()
            total_loss = 0
            for images,label in valid_loader:
                images = images.to(device)
                label = label.to(device)
                model_cnn.to(device)
                pred_ts=model_cnn(images)
                score_val = val_metric(pred_ts,label)
                val_loss = criterion(pred_ts, label)
                total_loss += val_loss.detach()
            avg_loss=total_loss/ len(train_loader)
            print('Val Loss:',avg_loss)
            val_score=val_metric.compute()
            print('CNN Validation Score:',val_score)
            writer.add_scalar("CNN Supervised F1/Validation", val_score, epoch)
            if avg_loss > last_loss:
                counter+=1
            else:
                counter=0

            last_loss = avg_loss
            if counter > 5:
                print('Early Stopping!')
                break
            else:
                if val_score > best_val:
                    best_val=val_score
                    print('Saving')
                    torch.save(model_cnn,
                        './CASS-CNN-part-ft.pt')
writer.flush()
```

```
Fine tunning Cov-T
Adjusting learning rate of group 0 to 3.0000e-04.
tensor([[-0.2224, -0.0796, -0.0395,  0.0047],
        [-0.2016, -0.0637, -0.0475, -0.0132],
        [-0.2842, -0.0978, -0.0022, -0.0042],
        [-0.1506, -0.0919,  0.0201, -0.0029],
        [-0.2032, -0.0587, -0.0537, -0.0144],
        [-0.1869, -0.0571, -0.0454, -0.0252],
        [-0.2704, -0.0947,  0.0181,  0.0343],
        [-0.2470, -0.0571, -0.0424, -0.0040],
        [-0.1797, -0.0699, -0.0362, -0.0193],
        [-0.2350, -0.0880,  0.0200, -0.0152],
        [-0.2367, -0.0918, -0.0361, -0.0323],
        [-0.1786, -0.0518, -0.0241, -0.0411],
        [-0.2378, -0.0537, -0.0488, -0.0198],
        [-0.2625, -0.1105,  0.0104, -0.0248],
        [-0.2353, -0.0843, -0.0563, -0.0045],
        [-0.1884, -0.0600, -0.0535, -0.0146]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 2.9713e-04.
tensor([[-0.2923, -0.1407, -0.3052, -0.2684],
        [-0.4407, -0.0278, -0.3867, -0.4032],
        [-0.2888, -0.1699, -0.3380, -0.2741],
        [-0.3111, -0.0465, -0.3216, -0.3530],
        [-0.3601, -0.1308, -0.3760, -0.3521],
        [-0.2935, -0.1751, -0.3503, -0.2644],
        [-0.4614, -0.0347, -0.4062, -0.4100],
        [-0.2804, -0.1245, -0.2536, -0.2571],
        [-0.4838, -0.0581, -0.4041, -0.3926],
        [-0.2956, -0.1631, -0.3425, -0.2655],
        [-0.2976, -0.1425, -0.3327, -0.2985],
        [-0.3182, -0.1158, -0.2790, -0.3249],
        [-0.2886, -0.1721, -0.3595, -0.2553],
        [-0.3173, -0.1108, -0.3267, -0.3305],
        [-0.3009, -0.1525, -0.2990, -0.3071],
        [-0.3857, -0.1401, -0.3280, -0.3435]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 2.8862e-04.
tensor([[-0.3948, -0.2500, -0.5073, -0.7663],
        [-0.3575, -0.2923, -0.4682, -0.7000],
        [-0.5710, -0.2074, -0.3536, -0.6390],
        [-0.1319, -0.1071, -0.1322, -0.2775],
        [-0.4152, -0.2180, -0.4753, -0.6081],
        [-0.3994, -0.2306, -0.5019, -0.7797],
        [-0.4137, -0.2330, -0.3641, -0.7165],
        [-0.3897, -0.1855, -0.4681, -0.5477],
        [-0.4533, -0.0918, -0.4000, -0.5921],
        [-0.4333, -0.1342, -0.3623, -0.5671],
        [-0.3304, -0.1288, -0.3039, -0.4880],
        [-0.4004, -0.2285, -0.4778, -0.7775],
        [-0.3670, -0.2833, -0.4688, -0.7029],
        [-0.4507, -0.2470, -0.3412, -0.8113],
        [-0.3658, -0.2500, -0.4569, -0.7508],
        [-0.3642, -0.1798, -0.3647, -0.6337]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 2.7480e-04.
tensor([[-0.2662, -0.2587, -0.3572, -0.4150],
        [-0.6134, -0.3329, -0.5444, -1.1773],
        [-0.2911, -0.1722, -0.2080, -0.4563],
        [-0.2964, -0.1690, -0.2964, -0.3847],
        [-0.4076, -0.2530, -0.4141, -0.7080],
        [-0.3821, -0.3358, -0.5522, -0.6768],
        [-0.3566, -0.2378, -0.4168, -0.4976],
        [-0.3342, -0.3496, -0.5732, -0.5608],
        [-0.5227, -0.3614, -0.5144, -0.9873],
        [-0.2169, -0.2244, -0.2937, -0.3260],
        [-0.4923, -0.1848, -0.5535, -0.5243],
        [-0.2264, -0.2831, -0.3964, -0.5277],
        [-0.2486, -0.2427, -0.3285, -0.3642],
        [-0.7652, -0.3903, -0.8725, -0.6416],
        [-0.4456, -0.2867, -0.5927, -0.4200],
        [-0.3068, -0.3001, -0.4632, -0.4359]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.]], device='cuda:0')
Adjusting learning rate of group 0 to 2.5621e-04.
tensor([[-0.6422, -0.5289, -0.3709, -0.9599],
        [-0.7582, -0.4959, -0.6721, -0.8641],
        [-0.4311, -0.1428, -0.3553, -0.5630],
        [-0.0638, -0.1564, -0.2281, -0.3715],
        [-0.4366, -0.4261, -0.5993, -0.3305],
        [-0.1529, -0.2014, -0.2697, -0.1098],
        [-0.2343, -0.2638, -0.3303, -0.1506],
        [-0.1839, -0.2544, -0.3481, -0.1777],
        [-0.0611, -0.1061, -0.1486, -0.1748],
        [-1.1402, -0.6159, -1.1525, -1.0170],
        [-0.5155, -0.3127, -0.2589, -0.8368],
        [-0.2496, -0.2741, -0.3732, -0.1878],
        [-0.0757, -0.1536, -0.2228, -0.2922],
        [-0.2297, -0.2646, -0.3817, -0.1796],
        [-0.1637, -0.1754, -0.2868, -0.1877],
        [-0.8618, -0.4387, -0.4267, -1.1588]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 2.3356e-04.
tensor([[-0.4038, -0.3359, -0.4730, -0.4515],
        [-0.3237, -0.1534, -0.1698, -0.5448],
        [-0.8480, -0.5261, -0.2865, -1.2341],
        [-0.3478, -0.3141, -0.3228, -0.4108],
        [-0.3441, -0.3420, -0.3961, -0.2814],
        [-0.2357, -0.2026, -0.1731, -0.3544],
        [-0.7444, -0.5353, -0.4136, -0.9286],
        [-0.4993, -0.3860, -0.4870, -0.3409],
        [-0.1973, -0.2060, -0.2667, -0.2216],
        [-0.1015, -0.1718, -0.1880, -0.1056],
        [-0.6544, -0.4917, -0.6224, -0.4747],
        [-0.3352, -0.3329, -0.4780, -0.4494],
        [-0.4437, -0.2688, -0.4379, -0.8660],
        [-0.7107, -0.3139, -0.6788, -0.8415],
        [-0.2931, -0.2786, -0.3978, -0.4460],
        [-0.3774, -0.3418, -0.4214, -0.2920]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 2.0771e-04.
tensor([[-1.0146, -0.5669, -0.4458, -1.4597],
        [-0.5474, -0.3786, -0.5102, -0.5783],
        [-0.4007, -0.3095, -0.3759, -0.3632],
        [-0.4278, -0.3577, -0.4131, -0.5275],
        [-0.5710, -0.3974, -0.5118, -0.6135],
        [-0.0802, -0.0758, -0.0750, -0.2546],
        [-0.5393, -0.4071, -0.4828, -0.5894],
        [-0.5271, -0.3050, -0.2762, -0.7357],
        [-0.4114, -0.1411, -0.2021, -0.6440],
        [-0.3631, -0.3198, -0.3641, -0.4156],
        [-0.6777, -0.2353, -0.0737, -1.2696],
        [-0.2637, -0.2440, -0.2894, -0.2542],
        [-0.3332, -0.1934, -0.3035, -0.5116],
        [-0.0375, -0.0900, -0.0204, -0.2607],
        [-0.5298, -0.4033, -0.4779, -0.5759],
        [-0.6551, -0.4546, -0.4906, -0.7768]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 1.7967e-04.
tensor([[-0.1954, -0.1015, -0.1275, -0.5064],
        [-0.6559, -0.1171, -0.2234, -0.9740],
        [-1.0307, -0.1828, -0.1082, -1.7341],
        [-0.2682, -0.3120, -0.3279, -0.3169],
        [-0.5701, -0.2959, -0.4474, -0.8382],
        [-1.0741, -0.5438, -0.5327, -1.3584],
        [-0.2371, -0.3312, -0.2913, -0.3634],
        [-0.3069, -0.1093, -0.2071, -0.6882],
        [-0.4237, -0.4506, -0.4636, -0.5345],
        [-0.0426, -0.0814, -0.0433, -0.2153],
        [-0.4425, -0.4178, -0.4726, -0.4404],
        [-0.5131, -0.4534, -0.5610, -0.5967],
        [-0.0946, -0.1357, -0.1033, -0.3411],
        [-0.4827, -0.4829, -0.5144, -0.5963],
        [-0.4654, -0.4295, -0.4879, -0.4433],
        [-0.0504, -0.1197, -0.0748, -0.3130]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 1.5050e-04.
tensor([[-0.8147,  0.0147, -0.1105, -1.2560],
        [-0.5488, -0.5010, -0.6521, -0.7018],
        [-0.5831, -0.1671,  0.0532, -0.9116],
        [-0.5021, -0.5172, -0.5843, -0.7184],
        [-0.4781, -0.2170, -0.1341, -0.8469],
        [-0.7260, -0.0667, -0.0690, -1.1165],
        [-0.4897, -0.4750, -0.5861, -0.5887],
        [-0.2363, -0.2867, -0.3132, -0.3231],
        [-0.1547, -0.2409, -0.1941, -0.4023],
        [-0.1012, -0.0556, -0.0894, -0.2675],
        [-0.3916, -0.4310, -0.4124, -0.5502],
        [-0.5128, -0.5456, -0.5825, -0.7599],
        [-0.1053, -0.1529, -0.1484, -0.3687],
        [-0.5304, -0.5468, -0.6187, -0.7519],
        [-0.2740, -0.3338, -0.3131, -0.4565],
        [-0.1527, -0.0781, -0.0912, -0.3977]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 1.2133e-04.
tensor([[-0.3009, -0.3422, -0.3332, -0.5277],
        [-0.0878, -0.1022, -0.0396, -0.3038],
        [-0.4094, -0.4742, -0.4441, -0.6921],
        [-0.9798, -0.5747, -0.4262, -1.2905],
        [-1.1801, -0.3237, -0.2914, -1.6629],
        [-0.0843, -0.1643, -0.1122, -0.3409],
        [-0.4739, -0.4916, -0.5056, -0.7233],
        [-0.1731, -0.0747, -0.1151, -0.4153],
        [-0.4048, -0.4824, -0.3873, -0.6428],
        [-0.1859, -0.1805, -0.1545, -0.3961],
        [-0.3615, -0.3705, -0.4047, -0.6114],
        [-0.3948, -0.4614, -0.4639, -0.6407],
        [-0.1006, -0.0693, -0.0693, -0.2788],
        [-0.4453, -0.0978, -0.0884, -0.8318],
        [-0.0790, -0.1470, -0.0795, -0.3562],
        [-0.6528, -0.5385, -0.5279, -0.8915]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 9.3289e-05.
tensor([[-0.1374, -0.0269, -0.1000, -0.2563],
        [-0.0865, -0.1649, -0.0539, -0.4144],
        [-0.1807, -0.0812, -0.1219, -0.3953],
        [-0.2622, -0.4203, -0.2699, -0.6356],
        [-0.3062, -0.3640, -0.3552, -0.5037],
        [-0.3381, -0.4253, -0.3874, -0.5685],
        [-0.4551, -0.4729, -0.5048, -0.6961],
        [-0.3411, -0.4182, -0.3851, -0.5648],
        [-0.1884, -0.0180, -0.1446, -0.3223],
        [-0.1995, -0.3463, -0.1714, -0.5304],
        [-1.6034, -0.2028,  0.2505, -1.9153],
        [-0.3166, -0.3997, -0.3692, -0.5492],
        [-0.0955, -0.1124, -0.1009, -0.2169],
        [-0.3886, -0.5367, -0.4242, -0.7818],
        [-0.0915, -0.1948, -0.0937, -0.4008],
        [-0.4176, -0.5067, -0.4131, -0.7585]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 6.7442e-05.
tensor([[-0.1990, -0.3076, -0.1006, -0.5152],
        [-0.1601, -0.2925, -0.2086, -0.4182],
        [-0.3686, -0.3743, -0.2338, -0.5606],
        [-0.0924, -0.1550, -0.0813, -0.3391],
        [-0.3815, -0.5660, -0.4609, -0.7333],
        [-1.4866,  0.2159,  0.4403, -1.4775],
        [-0.1940, -0.1562, -0.0764, -0.5155],
        [-0.1569, -0.2583, -0.1153, -0.4599],
        [-0.0704, -0.0840, -0.0641, -0.1996],
        [-0.2988, -0.4119, -0.3884, -0.4954],
        [-0.4909, -0.6463, -0.5370, -0.9076],
        [-0.3262, -0.4265, -0.4182, -0.5154],
        [-0.2090, -0.1532, -0.1042, -0.5515],
        [-0.8136, -0.5768, -0.3113, -1.1450],
        [-0.4758, -0.1712, -0.1707, -0.6477],
        [-0.3359, -0.4600, -0.4286, -0.5325]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.]], device='cuda:0')
Adjusting learning rate of group 0 to 4.4788e-05.
tensor([[-0.4530, -0.6902, -0.5477, -0.7560],
        [-0.4120, -0.6221, -0.4804, -0.7053],
        [-0.1139, -0.2419, -0.1485, -0.2699],
        [-0.1381, -0.1595, -0.1653, -0.2686],
        [-2.0856,  0.1742,  0.6327, -1.9835],
        [-0.0819, -0.0970, -0.0518, -0.2485],
        [-0.1074, -0.1916, -0.0839, -0.3751],
        [-0.7299, -0.3476, -0.1963, -0.8087],
        [-0.3813, -0.5804, -0.4513, -0.6626],
        [-0.0985, -0.1696, -0.0196, -0.4016],
        [-0.1255, -0.1384, -0.0445, -0.3698],
        [-0.1616, -0.2766, -0.1383, -0.4511],
        [-0.4815, -0.7125, -0.5580, -0.8034],
        [-0.2044, -0.1145, -0.1062, -0.4908],
        [-0.3546, -0.5682, -0.4481, -0.6151],
        [-0.0591, -0.1545,  0.0335, -0.3498]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 2.6195e-05.
tensor([[-0.3481, -0.5511, -0.3973, -0.6535],
        [-0.0419, -0.1719, -0.0447, -0.2825],
        [-0.2730, -0.4879, -0.3188, -0.5682],
        [-0.5257, -0.4573, -0.1879, -0.8555],
        [-0.7776,  0.0340,  0.2404, -0.7454],
        [-1.0547,  0.1269,  0.1809, -1.0437],
        [-0.3132, -0.4709, -0.3723, -0.5613],
        [-0.1455, -0.1174, -0.1667, -0.2343],
        [-0.9763, -0.4493, -0.1291, -1.1591],
        [-0.1450, -0.2616, -0.1411, -0.4108],
        [-0.2257, -0.3631, -0.2786, -0.4234],
        [-0.2315, -0.4356, -0.3222, -0.4656],
        [-0.1640, -0.1716, -0.0931, -0.4458],
        [-0.4408, -0.5763, -0.2967, -0.7218],
        [-0.1703, -0.1198, -0.1152, -0.3851],
        [-0.3044, -0.5193, -0.4044, -0.5728]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 1.2380e-05.
tensor([[-0.2925, -0.4910, -0.3724, -0.4327],
        [-0.2419, -0.4687, -0.2850, -0.4789],
        [-0.3166, -0.5342, -0.3497, -0.5781],
        [-1.1161, -0.1481,  0.3079, -1.0862],
        [-1.1491,  0.0448,  0.1583, -1.1058],
        [-0.1372, -0.1087, -0.1283, -0.2628],
        [-0.3570, -0.5991, -0.4141, -0.6266],
        [-0.1183, -0.1538, -0.0125, -0.4302],
        [-0.6987, -0.1695,  0.1049, -0.8009],
        [-0.1808, -0.1521, -0.0921, -0.4747],
        [-0.2499, -0.4641, -0.3110, -0.4559],
        [-0.3035, -0.4548, -0.3874, -0.4521],
        [-0.3059, -0.5209, -0.3386, -0.5684],
        [-0.1376, -0.1637, -0.0209, -0.4404],
        [-0.3112, -0.4738, -0.3936, -0.4589],
        [-0.3341, -0.4964, -0.4007, -0.5713]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 3.8726e-06.
tensor([[-0.3014, -0.5355, -0.3622, -0.5595],
        [-0.2108, -0.1844, -0.0611, -0.5712],
        [-0.3373, -0.4771, -0.4259, -0.4695],
        [-0.0911, -0.2048, -0.1102, -0.2839],
        [-0.6661, -0.1114,  0.1287, -0.7835],
        [-0.5317, -0.3320, -0.2602, -0.7224],
        [-0.3321, -0.4567, -0.4192, -0.4937],
        [-0.2232, -0.3641, -0.2729, -0.3461],
        [-0.7465, -0.1944, -0.0381, -0.8111],
        [-0.3116, -0.5638, -0.3472, -0.6145],
        [-0.2023, -0.1626, -0.1746, -0.4741],
        [-0.1286, -0.3266, -0.0864, -0.3496],
        [-0.2881, -0.5184, -0.3525, -0.5510],
        [-0.6287, -0.2186, -0.0480, -0.7565],
        [-0.9125, -0.0959,  0.2254, -0.9591],
        [-0.2803, -0.4822, -0.3428, -0.5016]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 1.0000e-06.
```

```
tensor([[-0.1273, -0.1887, -0.0583, -0.3432],
        [-0.1237, -0.1509, -0.0103, -0.3929],
        [-0.1374, -0.3011, -0.0486, -0.4031],
        [-0.1740, -0.3698, -0.2525, -0.3605],
        [-0.3132, -0.4906, -0.3971, -0.4372],
        [-0.4238, -0.5900, -0.5307, -0.5890],
        [-0.1815, -0.1709, -0.0993, -0.4247],
        [-0.2807, -0.5051, -0.3934, -0.4741],
        [-0.2320, -0.2809, -0.2833, -0.3584],
        [-0.6770, -0.7476, -0.6253, -0.9524],
        [-0.3283, -0.5641, -0.4129, -0.5688],
        [-0.5481, -0.2938, -0.1562, -0.6268],
        [-1.6029,  0.0236,  0.2321, -1.4923],
        [-0.1850, -0.3759, -0.2511, -0.3719],
        [-0.1515, -0.2381, -0.0828, -0.4171],
        [-0.0624, -0.1171, -0.0590, -0.1869]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 3.8726e-06.
tensor([[-0.2822, -0.5201, -0.3863, -0.5153],
        [-0.1823, -0.3402, -0.1311, -0.4321],
        [-0.1801, -0.3941, -0.2778, -0.3559],
        [-0.1412, -0.1264, -0.0486, -0.4002],
        [-0.2380, -0.1782, -0.0067, -0.4650],
        [-0.3101, -0.5490, -0.4030, -0.5605],
        [-0.4136, -0.3493, -0.1959, -0.6651],
        [-0.1610, -0.1313, -0.0667, -0.4357],
        [-0.2532, -0.4610, -0.3618, -0.4556],
        [-1.5881,  0.0971,  0.2893, -1.4738],
        [-0.3166, -0.3751, -0.4285, -0.4161],
        [-1.0002, -0.3313,  0.1397, -0.9838],
        [-0.2504, -0.4856, -0.3340, -0.4774],
        [-0.2346, -0.4718, -0.3474, -0.4397],
        [-0.1385, -0.1965, -0.1347, -0.3566],
        [-0.2878, -0.4760, -0.3944, -0.4984]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 1.2380e-05.
tensor([[-0.4349, -0.7397, -0.4661, -0.7607],
        [-0.0744, -0.0331, -0.0219, -0.1639],
        [-0.9574, -0.0479,  0.2423, -0.9186],
        [-0.3570, -0.5120, -0.3531, -0.6416],
        [-0.1849, -0.2613, -0.2383, -0.2959],
        [-0.1858, -0.0811, -0.1078, -0.3568],
        [-0.3798, -0.6523, -0.4467, -0.6504],
        [-1.2697, -0.0161,  0.1165, -1.2976],
        [-0.2543, -0.4237, -0.2938, -0.4536],
        [-0.1359, -0.0783, -0.1144, -0.2557],
        [-0.1174, -0.1784, -0.1445, -0.2043],
        [-0.3344, -0.6032, -0.3975, -0.5666],
        [-0.7686, -0.2535,  0.1037, -0.7998],
        [-0.2452, -0.4901, -0.2968, -0.4783],
        [-0.2532, -0.4418, -0.2398, -0.5237],
        [-0.3636, -0.6329, -0.4251, -0.6455]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 2.6195e-05.
tensor([[-0.6205, -0.1428, -0.0137, -0.6856],
        [-0.1404, -0.0926, -0.1733, -0.2263],
        [-0.1190, -0.2127, -0.1497, -0.2527],
        [-0.3730, -0.5711, -0.4724, -0.5815],
        [-0.1577, -0.1255, -0.1219, -0.3151],
        [-0.3488, -0.6178, -0.3718, -0.6582],
        [-1.2006,  0.0485,  0.2653, -1.0861],
        [-0.1440, -0.2562, -0.1231, -0.3858],
        [-0.7276, -0.2316,  0.1002, -0.8463],
        [-0.3690, -0.5344, -0.3848, -0.6893],
        [-0.3199, -0.6020, -0.4014, -0.6032],
        [-0.3520, -0.6374, -0.4146, -0.6706],
        [-0.3420, -0.3486, -0.3874, -0.4477],
        [-0.1070, -0.1580, -0.0939, -0.1755],
        [-0.5477, -0.4345, -0.0045, -0.8011],
        [-0.3435, -0.5579, -0.4351, -0.5732]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 4.4788e-05.
tensor([[-0.3956, -0.5612, -0.4493, -0.6909],
        [-0.2736, -0.4448, -0.4058, -0.3668],
        [-0.2786, -0.5273, -0.3743, -0.5161],
        [-0.2582, -0.1600, -0.2948, -0.3440],
        [-0.2794, -0.5619, -0.4053, -0.5177],
        [-0.8005, -0.5300, -0.2055, -0.9534],
        [-0.6363, -0.0589,  0.0403, -0.6999],
        [-0.2552, -0.4593, -0.1971, -0.5266],
        [-0.1498, -0.1879, -0.0984, -0.4161],
        [-0.3185, -0.5320, -0.3701, -0.6014],
        [-0.1597, -0.2278, -0.0755, -0.4745],
        [-0.9119, -0.0980,  0.0218, -0.9791],
        [-0.2023, -0.1169, -0.1311, -0.4444],
        [-0.1439, -0.2272, -0.1695, -0.2968],
        [-0.4317, -0.4942, -0.1588, -0.6353],
        [-0.9843, -0.2607, -0.0180, -1.0086]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 6.7442e-05.
tensor([[-0.0349, -0.1296, -0.0299, -0.1629],
        [-1.0541, -0.0146, -0.1697, -1.1310],
        [-0.1915, -0.4036, -0.2845, -0.4256],
        [-0.8344, -0.2199,  0.0999, -0.8881],
        [-1.0319, -0.0418, -0.0575, -1.0566],
        [-0.1479, -0.1658, -0.0351, -0.3665],
        [-0.6625, -0.2168,  0.0280, -0.8172],
        [-0.2735, -0.4798, -0.4249, -0.4055],
        [-0.6725, -0.7652, -0.3707, -0.9078],
        [-0.2827, -0.6083, -0.4251, -0.5353],
        [-0.1778, -0.3592, -0.2859, -0.3351],
        [-0.2685, -0.0997, -0.1936, -0.5005],
        [-0.1661, -0.2348, -0.1190, -0.4583],
        [-0.1773, -0.4261, -0.2579, -0.4189],
        [-0.3059, -0.6295, -0.4720, -0.5441],
        [-0.3275, -0.6230, -0.4400, -0.5523]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [1., 0., 0., 0.],
        [1., 0., 0., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 9.3289e-05.
tensor([[-0.2132, -0.4731, -0.3911, -0.3974],
        [-0.2177, -0.4793, -0.4087, -0.3939],
        [-0.7382, -0.2578,  0.0436, -0.9503],
        [-0.2982, -0.4800, -0.4692, -0.4781],
        [-0.1895, -0.3267, -0.3375, -0.2688],
        [-0.9063, -0.2158, -0.2576, -1.0832],
        [-0.0660, -0.0658, -0.0259, -0.1818],
        [-1.0683, -0.0504,  0.1315, -1.0531],
        [-0.2406, -0.4801, -0.3386, -0.5166],
        [-0.1971, -0.4706, -0.3832, -0.3727],
        [-0.1944, -0.2982, -0.2845, -0.4050],
        [-0.3249, -0.5923, -0.4891, -0.5846],
        [-0.7233, -0.3737,  0.0898, -0.9527]], device='cuda:0',
       grad_fn=<AddmmBackward0>)
tensor([[0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 1., 0., 0.],
        [0., 0., 0., 1.],
        [0., 0., 0., 1.],
        [0., 0., 1., 0.],
        [0., 0., 1., 0.]], device='cuda:0')
Adjusting learning rate of group 0 to 1.2133e-04.
{'train_loss': tensor(0.0761, device='cuda:0', grad_fn=<MeanBackward0>),
'Recall': tensor(0.0658), 'lr': 0.00012133399685858874}
Val Loss: tensor(0.1731, device='cuda:0')
CNN Validation Score: tensor(0.1816)
Saving
```

[35]:

```
model_vit=torch.load('cass-r50-vit-isic.pt')
#Train Correspong Supervised Vit
print('Fine tunning Cov-T')
model_vit.head=nn.Linear(in_features=768, out_features=4, bias=True)
criterion = FocalLoss(cfg.fl_alpha, cfg.fl_gamma)
metric = MyF1Score(cfg)
optimizer = torch.optim.Adam(model_vit.parameters(), lr = 3e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,T_max=cfg.
 ↪t_max,eta_min=cfg.min_lr,verbose=True)
model_vit.train()
val_metric=MyF1Score(cfg)
writer = SummaryWriter()
from torch.autograd import Variable
best=0
best_val=0
last_loss=math.inf
#Number of Epoches Change here
for epoch in range(1):
    for images,label in train_loader:
        model_vit.train()
        images = images.to(device)
        label = label.to(device)
        model_vit.to(device)
        pred_ts=model_vit(images)
        loss = criterion(pred_ts, label)
        score = metric(pred_ts,label)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        scheduler.step()
    train_score=metric.compute()
    logs = {'train_loss': loss, 'Recall': train_score, 'lr': optimizer.
 ↪param_groups[0]['lr']}
    writer.add_scalar("Supervised-ViT Loss/train", loss, epoch)
    writer.add_scalar("Supervised-ViT Recall/train", train_score, epoch)

    print(logs)
    if best < train_score:
        with torch.no_grad():
            best=train_score
            model_vit.eval()
            total_loss = 0
            for images,label in valid_loader:
                images = images.to(device)
                label = label.to(device)
                model_vit.to(device)
                pred_ts=model_vit(images)
                score_val = val_metric(pred_ts,label)
                val_loss = criterion(pred_ts, label)
                total_loss += val_loss.detach()
            avg_loss=total_loss/ len(train_loader)
            print('Val Loss:',avg_loss)
            val_score=val_metric.compute()
            print('ViT Validation Score:',val_score)
            writer.add_scalar("ViT Supervised F1/Validation", val_score, epoch)
            if avg_loss > last_loss:
                counter+=1
            else:
                counter=0

            last_loss = avg_loss
            if counter > 5:
                print('Early Stopping!')
                break
            else:
                if val_score > best_val:
                    best_val=val_score
                    print('Saving')
                    torch.save(model_vit,
                        './CASS-ViT-part-ft.pt')
writer.flush()
```

```
Fine tunning Cov-T
Adjusting learning rate of group 0 to 3.0000e-04.
Adjusting learning rate of group 0 to 2.9713e-04.
Adjusting learning rate of group 0 to 2.8862e-04.
Adjusting learning rate of group 0 to 2.7480e-04.
Adjusting learning rate of group 0 to 2.5621e-04.
Adjusting learning rate of group 0 to 2.3356e-04.
Adjusting learning rate of group 0 to 2.0771e-04.
Adjusting learning rate of group 0 to 1.7967e-04.
Adjusting learning rate of group 0 to 1.5050e-04.
Adjusting learning rate of group 0 to 1.2133e-04.
Adjusting learning rate of group 0 to 9.3289e-05.
Adjusting learning rate of group 0 to 6.7442e-05.
Adjusting learning rate of group 0 to 4.4788e-05.
Adjusting learning rate of group 0 to 2.6195e-05.
Adjusting learning rate of group 0 to 1.2380e-05.
Adjusting learning rate of group 0 to 3.8726e-06.
Adjusting learning rate of group 0 to 1.0000e-06.
Adjusting learning rate of group 0 to 3.8726e-06.
Adjusting learning rate of group 0 to 1.2380e-05.
Adjusting learning rate of group 0 to 2.6195e-05.
Adjusting learning rate of group 0 to 4.4788e-05.
Adjusting learning rate of group 0 to 6.7442e-05.
Adjusting learning rate of group 0 to 9.3289e-05.
Adjusting learning rate of group 0 to 1.2133e-04.
Adjusting learning rate of group 0 to 1.5050e-04.
Adjusting learning rate of group 0 to 1.7967e-04.
Adjusting learning rate of group 0 to 2.0771e-04.
Adjusting learning rate of group 0 to 2.3356e-04.
Adjusting learning rate of group 0 to 2.5621e-04.
Adjusting learning rate of group 0 to 2.7480e-04.
{'train_loss': tensor(0.0884, device='cuda:0', grad_fn=<MeanBackward0>),
'Recall': tensor(0.3158), 'lr': 0.0002748047070392304}
Val Loss: tensor(0.3427, device='cuda:0')
ViT Validation Score: tensor(0.2677)
Saving
```

10 Results

In this section, you should finish training your model training or loading your trained model. That is a great experiment! You should share the results with others with necessary metrics and figures.

Please test and report results for all experiments that you run with:

- specific numbers (accuracy, AUC, RMSE, etc)
- figures (loss shrinkage, outputs from GAN, annotation or label of sample pictures, etc)

[ ]:

```
# metrics to evaluate my model

# plot figures to better show the results

# it is better to save the numbers and figures for your presentation.
```

[153]:

```
class CFG:
    # data path
    data_path  = 'BrainMRI/testing.csv'
    train_imgs_dir  = 'BrainMRI/Testing'
    # model info
    # label info
    label_num2str = {0: 'glioma',
                     1: 'pituitary',
                     2:'notumor',
                     3:'meningioma'
                     }
    label_str2num = {'glioma': 0,
                     'pituitary':1,
                     'notumor':2,
                     'meningioma':3
                     }
    fl_alpha = 1.0  # alpha of focal_loss
    fl_gamma = 2.0  # gamma of focal_loss
    cls_weight = [0.2, 0.5970802919708029, 1.0, 0.25255474452554744] # copy the␣
↪cls_weight from previous step or just use the variable
    cnn_name='resnet50'
    vit_name='vit_base_patch16_384'
    seed = 77
    num_classes = 4
    batch_size = 16
    t_max = 16
    lr = 1e-3
    min_lr = 1e-6
    n_fold = 6
    num_workers = 8
    gpu_idx = 0
    device = torch.device(f'cuda:{gpu_idx}' if torch.cuda.is_available() else␣
↪'cpu')
    gpu_list = [gpu_idx]
```

[154]:

```
class Dataset(Dataset):
    def __init__(self, cfg, img_names: list, labels: list, transform=None):
        self.img_dir = cfg.train_imgs_dir
        self.img_names = img_names
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.img_names)

    def __getitem__(self, idx):
        img_path = self.img_names[idx]
        img = Image.open(img_path).convert('RGB')
        img_ts = self.transform(img)
        label_ts = self.labels[idx]
        return img_ts, label_ts
```

[155]:

```
class FocalLoss(nn.Module):
    """
    The focal loss for fighting against class-imbalance
    """
    def __init__(self, alpha=1, gamma=2):
        super(FocalLoss, self).__init__()
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = 1e-12  # prevent training from Nan-loss error
        self.cls_weights = torch.tensor([CFG.cls_weight],dtype=torch.float,␣
↪requires_grad=False, device=CFG.device)
```

```
    def forward(self, logits, target):
        """
        logits & target should be tensors with shape [batch_size, num_classes]
        """
        probs = torch.sigmoid(logits)
        one_subtract_probs = 1.0 - probs
        # add epsilon
        probs_new = probs + self.epsilon
        one_subtract_probs_new = one_subtract_probs + self.epsilon
        # calculate focal loss
        log_pt = target * torch.log(probs_new) + (1.0 - target) * torch.
 ↪log(one_subtract_probs_new)
        pt = torch.exp(log_pt)
        focal_loss = -1.0 * (self.alpha * (1 - pt) ** self.gamma) * log_pt
        focal_loss = focal_loss * self.cls_weights
        return torch.mean(focal_loss)
```

[156]:

```
"""
Define F1 score metric
"""
class MyF1Score(Metric):
    def __init__(self, cfg, threshold: float = 0.5, dist_sync_on_step=False):
        super().__init__(dist_sync_on_step=dist_sync_on_step)
        self.cfg = cfg
        self.threshold = threshold
        self.add_state("tp", default=torch.tensor(0), dist_reduce_fx="sum")
        self.add_state("fp", default=torch.tensor(0), dist_reduce_fx="sum")
        self.add_state("fn", default=torch.tensor(0), dist_reduce_fx="sum")

    def update(self, preds: torch.Tensor, target: torch.Tensor):
        assert preds.shape == target.shape
        preds_str_batch = self.num_to_str(torch.sigmoid(preds))
        target_str_batch = self.num_to_str(target)
        tp, fp, fn = 0, 0, 0
        for pred_str_list, target_str_list in zip(preds_str_batch,␣
 ↪target_str_batch):
            for pred_str in pred_str_list:
                if pred_str in target_str_list:
                    tp += 1
                if pred_str not in target_str_list:
                    fp += 1

            for target_str in target_str_list:
                if target_str not in pred_str_list:
                    fn += 1
        self.tp += tp
        self.fp += fp
        self.fn += fn

    def compute(self):
        #f1 = 2.0 * self.tp / (2.0 * self.tp + self.fn + self.fp)
        rec = self.tp/(self.tp + self.fn)
        return rec

    def num_to_str(self, ts: torch.Tensor) -> list:
        batch_bool_list = (ts > self.threshold).detach().cpu().numpy().tolist()
        batch_str_list = []
        for one_sample_bool in batch_bool_list:
            lb_str_list = [self.cfg.label_num2str[lb_idx] for lb_idx, bool_val␣
 ↪in enumerate(one_sample_bool) if bool_val]
            if len(lb_str_list) == 0:
                lb_str_list = ['healthy']
            batch_str_list.append(lb_str_list)
        return batch_str_list
```

[157]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
seed_everything(77)
cfg=CFG()
```

```
Global seed set to 77
```

We use the same Image Mean and STD as in the training steps

[158]:
```
DATASET_IMAGE_MEAN = (0.1857, 0.1857, 0.1858)
DATASET_IMAGE_STD = (0.2018, 0.2018, 0.2018)
```

[159]:
```
test_transform = tsfm.Compose([tsfm.Resize((384,384)),
                                tsfm.RandomApply([tsfm.ColorJitter(0.2, 0.2, 0.
 ↪2),tsfm.RandomPerspective(distortion_scale=0.2),], p=0.3),
                                tsfm.RandomApply([tsfm.ColorJitter(0.2, 0.2, 0.
 ↪2),tsfm.RandomAffine(degrees=10),], p=0.3),
                                tsfm.RandomVerticalFlip(p=0.3),
                                tsfm.RandomHorizontalFlip(p=0.3),
                                tsfm.ToTensor(),
                                tsfm.Normalize(DATASET_IMAGE_MEAN,␣
 ↪DATASET_IMAGE_STD), ])
```

[160]:
```
df=pd.read_csv('BrainMRI/testing.csv')
```

[161]:
```
df['target'].value_counts()
```

[161]:
```
notumor       405
meningioma    306
glioma        300
```

```
pituitary     300
Name: target, dtype: int64
```

[162]:
```
all_img_names: list = df['path'].values.tolist()
```

[163]:
```
len(all_img_names)
```

[163]:
```
1311
```

[164]:
```
all_img_labels_ts = []
for tmp_lb in df['target']:
    tmp_label = torch.zeros([CFG.num_classes], dtype=torch.float)
    label_num=CFG.label_str2num.get(tmp_lb)
    k=int(label_num)
    tmp_label[k] = 1.0
    all_img_labels_ts.append(tmp_label)
```

[165]:
```
test_dataset = Dataset(CFG, all_img_names,all_img_labels_ts, test_transform)
test_loader = DataLoader(test_dataset, batch_size=CFG.batch_size, shuffle=True,␣
 ↪num_workers=CFG.num_workers, drop_last=True)
```

[166]:
```
model=torch.load('./CASS-ViT-part-ft.pt')
```

[168]:
```
model.to(device)
```

[168]:
```
VisionTransformer(
  (patch_embed): PatchEmbed(
    (proj): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
    (norm): Identity()
  )
  (pos_drop): Dropout(p=0.0, inplace=False)
  (blocks): Sequential(
    (0): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (1): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (2): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (3): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (4): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (5): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (6): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
```

```
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (7): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (8): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (9): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (10): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
    )
    (11): Block(
      (norm1): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (attn): Attention(
        (qkv): Linear(in_features=768, out_features=2304, bias=True)
        (attn_drop): Dropout(p=0.0, inplace=False)
        (proj): Linear(in_features=768, out_features=768, bias=True)
        (proj_drop): Dropout(p=0.0, inplace=False)
      )
      (drop_path): Identity()
      (norm2): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
      (mlp): Mlp(
        (fc1): Linear(in_features=768, out_features=3072, bias=True)
        (act): GELU()
        (fc2): Linear(in_features=3072, out_features=768, bias=True)
        (drop): Dropout(p=0.0, inplace=False)
      )
```

```
      )
    )
    (norm): LayerNorm((768,), eps=1e-06, elementwise_affine=True)
    (pre_logits): Identity()
    (head): Linear(in_features=768, out_features=4, bias=True)
  )
```

[169]:

```
criterion = FocalLoss(cfg.fl_alpha, cfg.fl_gamma)
metric = MyF1Score(cfg)
val_metric=MyF1Score(cfg)
optimizer = torch.optim.Adam(model.parameters(), lr = 3e-4)
scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(optimizer,T_max=cfg.
 ↪t_max,eta_min=cfg.min_lr,verbose=True)
model.eval()

with torch.no_grad():
    for images,label in test_loader:
        images = images.to(device)
        label = label.to(device)
        model.to(device)
        pred_ts=model(images)
        loss = criterion(pred_ts, label)
        score = metric(pred_ts, label)
test_score=metric.compute()
logs = {'train_loss': loss, 'Recall': test_score, 'lr': optimizer.
 ↪param_groups[0]['lr']}
print(logs)
```

```
Adjusting learning rate of group 0 to 3.0000e-04.
```

```
/home/hdeng11/.local/lib/python3.10/site-
packages/torchmetrics/utilities/prints.py:36: UserWarning: Torchmetrics v0.9
introduced a new argument class property called `full_state_update` that has
                not been set for this class (MyF1Score). The property determines
if `update` by
                default needs access to the full metric state. If this is not
the case, significant speedups can be
                achieved and we recommend setting this to `False`.
                We provide an checking function
                `from torchmetrics.utilities import check_forward_no_full_state`
                that can be used to check if the `full_state_update=True` (old
and potential slower behaviour,
                default for now) or if `full_state_update=False` can be used
safely.

  warnings.warn(*args, **kwargs)
```

```
{'train_loss': tensor(0.2133, device='cuda:0'), 'Recall': tensor(0.3094), 'lr':
```

```
0.0003}
```

10.1 Model comparison

[2]:

```
# compare you model with others
# you don't need to re-run all other experiments, instead, you can directly␣
 ↪refer the metrics/numbers in the paper

#Not applicable in draft since we are running with only one epoche
```

11 Discussion

We set up a local test environment to test whether our CASS implementation was capable of running successfully for a single epoch with a batch size of 16. This was successful and a single epoch took about 30 minutes using a GPU with 22GB RAM. After training for one epoch on the Brain MRI dataset, we achieved a Recall value of 0.3 which is only slightly better than a naive random baseline. This was to be expected given the extremely limited training duration and only served as a sanity check that our implementation was functioning correctly. We believe this is a strong indicator that this paper is reproducible.

Our next step is to scale up our training process and run the model for a sufficient number of epochs (50+) in order to meaningfully compare our results with those reported by the authors. This is a potential challenge given that our GPU has less than half of the memory of the GPU used by the authors. This will likely restrict our batch size to less than 20, which is not ideal. To address this issue, we're considering setting up a cloud environment with more powerful GPUs so that we can reduce the overall training time and use a larger batch size. We will assess the feasibility and cost effectiveness of that option in the coming days and make a decision based on the available resources and time constraints.

We also plan to evaluate using other relevant metrics, like F1 score. If time permits, we may extend our study to another dataset, that being the DermoFIT dataset which is a private, paid dataset. We have applied for access and are still waiting for a decision.

12 References

1. Singh, P. and Cirrone, J., "Efficient Representation Learning for Healthcare with Cross-Architectural Self-Supervision", arXiv e-prints, 2023.

13 Feel free to add new sections