COMP197 Group Project

> Instruction.pdf is provided here, but it is more recommended to read the instructions in readme.md file.

Installation

The **most recommended** installation is to first create a conda environment and then run the following command to install the essential packages:

- for CPU developer:

```
conda install pytorch torchvision cpuonly -c pytorch
```

- for CUDA developer, please check the CUDA version of your system:

```
conda install pytorch torchvision pytorch-cuda=12.1 -c pytorch -c nvidia
```

or

```
conda install pytorch torchvision pytorch-cuda=11.8 -c pytorch -c nvidia
```

Extra Packages

In these project, we adopted exactly 3 extra pip packages for our development:

- *segmentation-models-pytorch*
- *scipy*
- *opencv-python*

Please install them via pip (not supported by conda) by running the following command:

```
pip install -r requirements.txt
```

Datasets

Oxford-IIIT Pet Dataset

Run the following command in project root directory:

```
python data/download.py
```

then the dataset will appear in `data/oxford-iiit-pet`.

Cats-vs-Dogs Dataset

Click here to Download the Whole `PetImages` Folder from the OneDrive as a zip file, then extract it to `data` directory.

ImageNet-1k-valid

Click here to Download `ILSVRC2012_devkit_t12.tar.gz` and `ILSVRC2012_img_val.tar.gz`, then place them in `data` directory.

Execution

This guide will provide you with the instruction to run this project and reproduce the result we obtained in our report. Including the method of training the pre-traine model, fine-tune model and evaluating the fine-tune model. However, this code repository does not contain the plot and visualization code that produce the figures in the report due to the limited 3 extra packages we can use in this project.

Also, the saved model files are not included in this repository, you can either train the model from scratch or download the saved model files from the ***OneDrive (Hyperlink)***. Models are placed in the `checkpoints` directory, inside the `checkpoints` directory, pre-trained models are placed in the `01-pre-trained` directory, and fine-tuned models are placed in the `02-fine-tuned` directory. Also, if you want to test baseline models, they are placed in the `03-baseline-vit` directory.

During the model training, we trained different models with different configurations and manually configured the parameters in the code. Almost all the configurable parameters are placed in `settings.py` file, but for some specific configurations you will need to modify from the target file.

Training Pre-train Models for Encoder

As explained in our report, the segmentation model is composed of two parts: encoder and decoder. The encoder part is the ViT encoders blocks of the model, pre-trained on ImageNet-1k dataset, trained by predicting the masked patches in the image. The decoder model in the pretraining is also blocks of ViT, but the decoder blocks here are only used for training the encoder blocks, and not utilized in the final segmentation model.

To train the encoder model, you can run the following command in the project root directory:

```
python pre-train.py
```

The default Dataset is *"ImageNet"* and you may set this to *"KaggleCatsAndDogs"* in `settings.py` file. You can also feel free to change the batch size based your GPU memory to accelerate the training process, and your target number of epochs. And you can configure other parameters in the `settings.py` file, such as the number of epochs, saving frequency, and batch size. All the configurable parameters are placed in the `settings.py` file starting with the prefix `PRE_TRAINING_`.

When the training is finished, you can find the trained model saved in the `models/checkpoints/MaskedAutoencoderViT/<tim` directory, where `<time_stamp>` is the time when the training started. The model will be saved as `epoch_<epoch_number>.pth`, where `<epoch_number>` is the number of epochs the model has been trained.

Training Fine-tune Models for Segmentation

If you do not want to train a baseline segmentation model, you will need to finish the pre-training of the encoder model first. Then when you have a saved encoder model, you will first need to configure the `settings.py` file to set the `PRETRAINED_MODEL` to the path of the saved encoder model.

For example:

```
PRE_TRAINED_MODEL = './models/checkpoints/MaskedAutoencoderViT/2024-04-13_16-12-13/epoch_1.pt'
```

After that, you can run the following command in the project root directory to train the fine-tune model:

```
python fine-tune.py
```

The model will be trained on the Oxford-IIIT Pet dataset, and this is the only dataset we used for training the segmentation model.

By default configuration, this script will train the model for 60 epochs, and save the model in the `models/checkpoints/VitEncodedUnet/<time_stamp>` directory every 10 epochs, with a batch size of 50. The model will be saved as `epoch_<epoch_number>.pth` which is same format as the pre-training model.

You can configure the batch size, number of epochs, and other parameters in the `settings.py` file with editing the Constants starting with the prefix `FINE_TUNING_`.

Evaluating Fine-tune Models

After finishing the training of the fine-tune model, you can evaluate the model performance including the binary accuracy, dice loss, dice score. But before that, just like the training process, you will need to configure the `settings.py` file to set the `FINE_TUNED_MODEL` to the path of the saved fine-tune model.

For example:

```
FINE_TUNED_MODEL = './models/checkpoints/ViTEncodedUnet/2024-04-13_16-34-48/epoch_1.pt'
```

Then you can run the following command in the project root directory to evaluate the model:

```
python test.py
```

The script will load the model and evaluate the model on the holdout test dataset of the Oxford-IIIT Pet dataset. The evaluation results will be printed in the console, including the binary accuracy, dice loss, and dice score.

More Configurations

In case that you may want to reproduce the more results in the report, including training the baseline model, different datasets and different pre-processing methods, more congirable parameters will be introduced below.

Use Baseline Model

A baseline model in this project is defined as a model that is trained from scratch without pre-training the encoder model. To train the baseline model, you can set the `BASELINE_MODE` to `True` in the `settings.py` file, then the model will ignore the pre-trained encoder model parameters and train the segmentation model from scratch.

```
BASELINE_MODE = True # Default is False
```

Configure Datasets

As mentioned above, the pre-training model can be trained on the ImageNet-1k dataset or the Kaggle Cats and Dogs dataset. The fine-tune model can only be trained on the Oxford-IIIT Pet dataset. Hence that the dataset of pre-training is configurable in the `settings.py` file, there are two acceptable values: `ImageNet` and `KaggleCatsAndDogs` for the `DATASET` constant.

```
DATASET = 'ImageNet' # `ImageNet` or `KaggleCatsAndDogs`
```

Configure Pre-processing

In the report, we have also done some experiments that horizontally compared the performance of the model with different pre-processing methods in pre-training and fine-tuning stages. By default they are disabled in the settings, but you can enable them by uncommenting the corresponding functions in the `transforms.Compose` function in the `settings.py` file.

In the pre-training stage, you can enable gaussian blur by setting the `PRE_TRAINING_BLURRING` to an odd value (3, 5, 7, 9) other than 1 to enable the gaussian blur in the pre-processing pipeline. The value is the gaussian kernel size, and the default value is 1 which means the gaussian blur is disabled.

```
PRE_TRAINING_BLURRING = 3 # [1, 3, 5, 7, 9]
```

And in the fine-tuning stage, you can enable the edge enhancement in the pre-processing pipeline by setting the `FINE_TUNING_TRANSFORM` by uncommenting the `CannyEdgeDetection(100, 200),` function in the `transforms.Compose` function.

```
FINE_TUNING_TRANSFORMS = Compose([
    CannyEdgeDetection(100, 200),
    Preprocess(),
])
```