Predictive Exit: Prediction of Fine-Grained Early Exits for Computation- and Energy-Efficient Inference

Required Packages

torch, torchvision, argparse, numpy.

File Description

- `nikolaos/`
  - `bof_utils.py`
    the data structure of BoF (Bag-of-Features) pooling
- `global_param.py`
  network initializations, the training and evaluation hyperparameters, the hardware configurations and so on
- `inference.py`
  inference of trained neural networks
- `main_new.py`
  the main program
- `models_1.py`
  the different types of neural network models including vgg19 for cifar10, cifar100
- `models_2.py`
  the different types of neural network models including vgg19 for stl10, svhn
- `models_3.py`
  the different types of neural network models including Resnet-34
- `train.py`
  training of neural networks
- `utils_new.py`
  some general functions
- `create_custom_dataloader`
  the functions to create customized dataloaders
- `generate_scripts.py`
  the python program to generate shell scripts for testings automatically. Except for only very few cases, this file is useless. Feel free to delete it
- `README.md`
  this file

Run the Codes

The overal Commands

The command looks like the followin
`python main_new.py --model_name MODEL_NAME --pretrained_file PRETRAINED_FILE_NAME --optimizer OPTIM_NAME --train_mode TRAIN_MODE --stat_each_layer STAT --evaluate_mode EVALUATE_MODE --task TASK --device DEVICE --trained_file_suffix SUFFIX --beta BETA --save SAVE --dataset_type DATASET --jump 1 --quantization 0 --forward_mode FORWARD_MODE --start_layer START_LAYER`
Their meanings are specified as below. Note that not all arguments are meaningful at the same time, for example, when the `TASK` is 'train', then `EVALUATE_MODE` becomes meaningless. Actually you don't need to include all these arguments in the command. Every option has a default argument, which is specified in the `main_new.py`, and when an option is missing in the command, the default value would be used.

- `MODEL_NAME`: the name of the model, either 'vgg19' or 'Resnet34'.
- `PRETRAINED_FILE_NAME`: the file that stores the pretrained model. They are all stored under the directory 'models_new'
    - '[model_name]_normal_default.pt' is the model for vanilla training
    - '[model_name]_original_default.pt' is the model with early-exit layers implemented but only original layers pre-trained.
    - '[model_name]_exits_default.pt' means the model with early-exit layers implemented and well trained.
- `OPTIM_NAME`: the name of the optimizer. Either 'sgd' or 'adam'. Usually by using 'adam' the training can be much more efficient.
- `TRAIN_MODE`: the mode for training.
    - 'original': train the model with early-exit layers implemented but only train the original layers and ignore the early-exit layers. This mode stores a pre-trained file that is necessary for the following 'exits' mode. This type can also be treated as a normal model without any exit layer.
    - 'exits': train the early-exit layers of the model, assuming that the original layers have already been trained.
- `STAT`: whether to collect the statistics of each layer, 1 for True and 0 for False
- `EVALUATE_MODE`: the mode for evaluation.
    - 'normal': evaluate the normal mode without any implementation of early exit layers
    - 'exits': evaluate the model with early exit layers and adaptive inference methods
- `TASK`: the task type, either 'train' or 'evaluate'. 'train' means to train and store a model based on training dataset, while 'evaluate' means to inference a model based on testing dataset
- `DEVICE`: the device where the training and inference is executed. Either 'cpu' or 'cuda'
- `SUFFIX`: when `SAVE` is non-zero, the model would be saved in a .pt file whose name consists of some some setting parameters and the string `SUFFIX`. This option is set mainly used to differentiate files under the same hyperparameters, so that they do not overwrite each other.
- `BETA`: the hyperparameter to control the tradeoff between accuracy and speed-up in early-exit scheme. Refer to the paper or our report for more details
- `SAVE`: if non-zero, then save the model, otherwise does not save the model

- `dataset_type`: cifar10, cifar100, stl10, svhn. When running stl10 in the Resnet, another pooling layer is required since its images are larger, so you need to modify one line code in line 20 of `model_news_3.py`. When running cifar100 in the vgg19, you may also need to change one line code that change a variable named `datatype` from 10 to 100.
- `jump`: no use and set it as 1
- `quantization`: 0 means `Floart 32` and 1 means `int8`
- `forward_mode`: either `normal_forward` or `accuracy_forward`. `normal_forward` means the program will run normal and `accuracy_forward` means the program start to test the prediction accuracy based on the start layer.
- `start_layer`: a number, the start layer,

Changing the Experimental Setting

we can change the setting of hyperparameters by making corresponding changes in `global_param.py`. For example, `vgg19_original_train_hyper` (line 408 in `global_param.py`) specifies the hyperparameter for normal training of cifar model. By changing the epoch number, batch size and so on, the hyperparameters for that task will be changed accordingly.

Changing the Model Structure

Unfortunately, if you want to modify the structure for a neural network model (e.g. add a layer or enlarge the layer) or define a new structure, you'll need to do that by yourself by directly modifying the file `models.py`. You would probably need to modify (or create) their layer name lists, their initialization and hyperparameter dictionaries in `global_param.py` accordingly. That would be somewhat frustrating, so be prepared.

Examples

Trainings

!!The training should be done on the computer / server.

The first thing you'll need to do is to train a network. Say you want to train the network model `vgg19` with early exit layers, you will first train the normal layers by executing using cifar10 as an example:

```
python3 main_new.py --model_name vgg19 --optimizer adam --train_mode original --
task train --device cuda --trained_file_suffix cifar10 --save 1 --dataset_type
cifar10 --jump 1
```

After that, you can start training the early exit layers by executing the command:

```
python3 main_new.py --model_name vgg19 --optimizer adam --train_mode exits --
task train --device cuda --pretrained_file autodl-
tmp/vgg19_train_original_cifar10.pt --trained_file_suffix cifar10 --save 1 --
dataset_type cifar10 --jump 1
```

Now basically you've done the training, and the trained model would be stored in `autodl-tmp/vgg19_train_exits_cifar10.pt`. And from the command output, you'll be able to see the accuracy after the training. If you are not satisfied with the accuracy, you could modify the model structures or hyperparameters and then do the above steps again.

Inferences

After training the model, you may want to test the early exit results by doing some inferences with different values of the parameter `beta`. There are two set of forward mode and `start_layer` that you need to input:

`normal_forward`: Run the inference with prediction engine

`accuracy_forward`: Test the accuracy of the prediction engine

`start_layer`: Choose the L_0

Then the command you're gonna type in the shell would be like:

```
python3 main_new.py --model_name vgg19 --optimizer adam --train_mode exits --
task evaluate --device cuda --pretrained_file autodl-
tmp/vgg19_train_exits_cifar10.pt  --beta 15 --save 0 --dataset_type cifar10 --
evaluate_mode exits --jump 1 --quantization 0 --forward_mode normal_forward --
start_layer 8
```