Homework 4

Instructions

- This homework focuses on understanding and applying CoCoOp for CLIP prompt tuning. It consists of **four questions** designed to assess both theoretical understanding and practical application.
- Please organize your answers and results for the questions below and submit this jupyter notebook as **a .pdf file**.
- **Deadline: 11/26 (Sat) 23:59**

Preparation

- Run the code below before proceeding with the homework (Q1, Q2).
- If an error occurs, click 'Run Session Again' and then restart the runtime from the beginning.

[ ] ↳ *숨겨진 셀 1개*

Q1. Understanding and implementing CoCoOp

- We have learned how to define CoOp in Lab Session 4.
- The main difference between CoOp and CoCoOp is **meta network** to extract image tokens that is added to the text prompt.
- Based on the CoOp code given in Lab Session 4, fill-in-the-blank exercise to test your understanding of critical parts of the CoCoOp.

```python
import torch.nn as nn

class CoCoOpPromptLearner(nn.Module):
    def __init__(self, cfg, classnames, clip_model):
        super().__init__()
        n_cls = len(classnames)
        n_ctx = cfg.TRAINER.COCOOP.N_CTX
        ctx_init = cfg.TRAINER.COCOOP.CTX_INIT
        dtype = clip_model.dtype
        ctx_dim = clip_model.ln_final.weight.shape[0]
        vis_dim = clip_model.visual.output_dim
        clip_imsize = clip_model.visual.input_resolution
        cfg_imsize = cfg.INPUT.SIZE[0]
        assert cfg_imsize == clip_imsize, f"cfg_imsize ({cfg_imsize}) must equal to clip_imsize ({clip_imsize})"

        if ctx_init:
            # use given words to initialize context vectors
            ctx_init = ctx_init.replace("_", " ")
            n_ctx = len(ctx_init.split(" "))
            prompt = clip.tokenize(ctx_init)
            with torch.no_grad():
                embedding = clip_model.token_embedding(prompt).type(dtype)
            ctx_vectors = embedding[0, 1: 1 + n_ctx, :]
            prompt_prefix = ctx_init
        else:
            # random initialization
            ctx_vectors = torch.empty(n_ctx, ctx_dim, dtype=dtype)
            nn.init.normal_(ctx_vectors, std=0.02)
            prompt_prefix = " ".join(["X"] * n_ctx)

        print(f'Initial context: "{prompt_prefix}"')
        print(f"Number of context words (tokens): {n_ctx}")

        self.ctx = nn.Parameter(ctx_vectors)  # Wrap the initialized prompts above as parameters to make them trainable.

        ### Tokenize ###
        classnames = [name.replace("_", " ") for name in classnames]  # 예) "Forest"
        name_lens = [len(_tokenizer.encode(name)) for name in classnames]
        prompts = [prompt_prefix + " " + name + "." for name in classnames] # 예) "A photo of Forest."

        tokenized_prompts = torch.cat([clip.tokenize(p) for p in prompts]) # 예) [49406, 320, 1125, 539...]



        ####################################
        ####### Q1. Fill in the blank #######
        ########## Define Meta Net ##########
        self.meta_net = nn.Sequential(OrderedDict([
            #("linear1", "fill in here"(vis_dim, vis_dim // 16)),
            ("linear1", nn.Linear(vis_dim, vis_dim // 16)),
```

```
            ("relu", nn.ReLU(inplace=True)),
            ("linear2", nn.Linear(vis_dim // 16, ctx_dim))
        ]))
        ###################################
        ## Hint: meta network is composed to linear layer, relu activation, and linear layer.



        if cfg.TRAINER.COCOOP.PREC == "fp16":
            self.meta_net.half()

        with torch.no_grad():
            embedding = clip_model.token_embedding(tokenized_prompts).type(dtype)

        # These token vectors will be saved when in save_model(),
        # but they should be ignored in load_model() as we want to use
        # those computed using the current class names
        self.register_buffer("token_prefix", embedding[:, :1, :])  # SOS
        self.register_buffer("token_suffix", embedding[:, 1 + n_ctx:, :])  # CLS, EOS
        self.n_cls = n_cls
        self.n_ctx = n_ctx
        self.tokenized_prompts = tokenized_prompts  # torch.Tensor
        self.name_lens = name_lens

    def construct_prompts(self, ctx, prefix, suffix, label=None):
        # dim0 is either batch_size (during training) or n_cls (during testing)
        # ctx: context tokens, with shape of (dim0, n_ctx, ctx_dim)
        # prefix: the sos token, with shape of (n_cls, 1, ctx_dim)
        # suffix: remaining tokens, with shape of (n_cls, *, ctx_dim)

        if label is not None:
            prefix = prefix[label]
            suffix = suffix[label]

        prompts = torch.cat(
            [
                prefix,  # (dim0, 1, dim)
                ctx,  # (dim0, n_ctx, dim)
                suffix,  # (dim0, *, dim)
            ],
            dim=1,
        )

        return prompts

    def forward(self, im_features):
        prefix = self.token_prefix
        suffix = self.token_suffix
        ctx = self.ctx  # (n_ctx, ctx_dim)



        ############################################
        ######### Q2,3. Fill in the blank #########
        #bias = self.meta_net("Fill in here, Hint: Image feature is given as input to meta network")  # (batch, ctx_dim)
        bias = self.meta_net(im_features)  # (batch, ctx_dim)
        bias = bias.unsqueeze(1)  # (batch, 1, ctx_dim)
        ctx = ctx.unsqueeze(0)  # (1, n_ctx, ctx_dim)
        #ctx_shifted = ctx + "Fill in here, Hint: Add meta token to context token"  # (batch, n_ctx, ctx_dim)
        ctx_shifted = ctx + bias  # (batch, n_ctx, ctx_dim)
        ############################################
        ############################################



        # Use instance-conditioned context tokens for all classes
        prompts = []
        for ctx_shifted_i in ctx_shifted:
            ctx_i = ctx_shifted_i.unsqueeze(0).expand(self.n_cls, -1, -1)
            pts_i = self.construct_prompts(ctx_i, prefix, suffix)  # (n_cls, n_tkn, ctx_dim)
            prompts.append(pts_i)
        prompts = torch.stack(prompts)

        return prompts


class CoCoOpCustomCLIP(nn.Module):
    def __init__(self, cfg, classnames, clip_model):
        super().__init__()
        self.prompt_learner = CoCoOpPromptLearner(cfg, classnames, clip_model)
        self.tokenized_prompts = self.prompt_learner.tokenized_prompts
        self.image_encoder = clip_model.visual
        self.text_encoder = TextEncoder(clip_model)
```

```
        self.logit_scale = clip_model.logit_scale
        self.dtype = clip_model.dtype

    def forward(self, image, label=None):
        tokenized_prompts = self.tokenized_prompts
        logit_scale = self.logit_scale.exp()

        image_features = self.image_encoder(image.type(self.dtype))
        image_features = image_features / image_features.norm(dim=-1, keepdim=True)


        ##########################################
        ######### Q4. Fill in the blank #########
        #prompts = self.prompt_learner("Fill in here")
        prompts = self.prompt_learner(image_features)
        ##########################################
        ##########################################


        logits = []
        for pts_i, imf_i in zip(prompts, image_features):
            text_features = self.text_encoder(pts_i, tokenized_prompts)
            text_features = text_features / text_features.norm(dim=-1, keepdim=True)
            l_i = logit_scale * imf_i @ text_features.t()
            logits.append(l_i)
        logits = torch.stack(logits)

        if self.prompt_learner.training:
            return F.cross_entropy(logits, label)

        return logits
```

Q2. Trainining CoCoOp

In this task, you will train CoCoOp on the EuroSAT dataset. If your implementation of CoCoOp in Question 1 is correct, the following code should execute without errors. Please submit the execution file so we can evaluate whether your code runs without any issues.

```
# Train on the Base Classes Train split and evaluate accuracy on the Base Classes Test split.
args.trainer = "CoCoOp"
args.train_batch_size = 4
args.epoch = 100
args.output_dir = "outputs/cocoop"

args.subsample_classes = "base"
args.eval_only = False
cocoop_base_acc = main(args)
```

```
epoch [93/100] batch [20/20] time 0.100 (0.132) data 0.000 (0.018) loss 0.1763 (0.2449) lr 3.9271e-05 eta 0:00:18
epoch [94/100] batch [20/20] time 0.144 (0.152) data 0.000 (0.019) loss 0.2859 (0.2261) lr 3.0104e-05 eta 0:00:18
epoch [95/100] batch [20/20] time 0.100 (0.181) data 0.000 (0.028) loss 0.1564 (0.1853) lr 2.2141e-05 eta 0:00:18
epoch [96/100] batch [20/20] time 0.098 (0.138) data 0.000 (0.018) loss 0.4089 (0.1330) lr 1.5390e-05 eta 0:00:11
epoch [97/100] batch [20/20] time 0.097 (0.130) data 0.000 (0.019) loss 0.0698 (0.1542) lr 9.8566e-06 eta 0:00:07
epoch [98/100] batch [20/20] time 0.101 (0.129) data 0.000 (0.019) loss 0.2188 (0.2041) lr 5.5475e-06 eta 0:00:05
epoch [99/100] batch [20/20] time 0.143 (0.176) data 0.000 (0.016) loss 0.0691 (0.1264) lr 2.4666e-06 eta 0:00:03
epoch [100/100] batch [20/20] time 0.106 (0.132) data 0.000 (0.027) loss 0.0025 (0.1101) lr 6.1680e-07 eta 0:00:00
Checkpoint saved to outputs/cocoop/prompt_learner/model.pth.tar-100
Finish training
Deploy the last-epoch model
Evaluate on the *test* set
100%|██████████| 42/42 [01:04<00:00,  1.53s/it]=> result
* total: 4,200
* correct: 3,813
* accuracy: 90.8%
* error: 9.2%
* macro_f1: 90.9%
Elapsed: 0:06:21
```

```
# Accuracy on the New Classes.
args.model_dir = "outputs/cocoop"
args.output_dir = "outputs/cocoop/new_classes"
args.subsample_classes = "new"
args.load_epoch = 100
args.eval_only = True
coop_novel_acc = main(args)
```

```
Loading trainer: CoCoOp
Loading dataset: EuroSAT
Reading split from /content/ProMetaR/data/eurosat/split_zhou_EuroSAT.json
Loading preprocessed few-shot data from /content/ProMetaR/data/eurosat/split_fewshot/shot_16-seed_1.pkl
SUBSAMPLE NEW CLASSES!
Building transform_train
+ random resized crop (size=(224, 224), scale=(0.08, 1.0))
+ random flip
+ to torch tensor of range [0, 1]
+ normalization (mean=[0.48145466, 0.4578275, 0.40821073], std=[0.26862954, 0.26130258, 0.27577711])
Building transform_test
+ resize the smaller edge to 224
+ 224x224 center crop
+ to torch tensor of range [0, 1]
+ normalization (mean=[0.48145466, 0.4578275, 0.40821073], std=[0.26862954, 0.26130258, 0.27577711])
---------  -------
Dataset    EuroSAT
# classes  5
# train_x  80
# val      20
# test     3,900
---------  -------
Loading CLIP (backbone: ViT-B/16)
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:617: UserWarning: This DataLoader will create 8 worker processes in total
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr
  warnings.warn(
/content/ProMetaR/dassl/utils/torchtools.py:102: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value)
  checkpoint = torch.load(fpath, map_location=map_location)
Building custom CLIP
Initial context: "a photo of a"
Number of context words (tokens): 4
Turning off gradients in both the image and the text encoder
Parameters to be updated: {'prompt_learner.ctx', 'prompt_learner.meta_net.linear1.weight', 'prompt_learner.meta_net.linear2.bias', 'prompt_learne
Loading evaluator: Classification
Loading weights to prompt_learner from "outputs/cocoop/prompt_learner/model.pth.tar-100" (epoch = 100)
Evaluate on the *test* set
100%|██████████| 39/39 [00:59<00:00,  1.53s/it]=> result
* total: 3,900
* correct: 1,687
* accuracy: 43.3%
* error: 56.7%
* macro_f1: 39.0%
```

Q3. Analyzing the results of CoCoOp

Compare the results of CoCoOp with those of CoOp that we trained in Lab Session 4. Discuss possible reasons for the performance differences observed between CoCoOp and CoOp.

결과를 보면, base 클래스에 대해 coOp은 정확도 91.4%, cocoOp은 정확도 90.8%로 둘 다 충분히 높은 성능을 보여줬고, new 클래스에 대해 coOp은 51.5%, cocoOp은 43.3%로 coOp이 더 나은 성능을 보여줬다. 이 결과는 new 클래스에 대해 cocoOp이 더 나은 성능을 보일 것이라고 기대한 바

와는 상반된 결과이다. 두 모델의 가장 큰 차이는 컨텍스트 토큰에 있는데, coOp은 16개의 충분한 토큰을 사용한 반면, cocoOp은 4개의 적은 토큰을 사용했다. cocoOp은 4개의 토큰만으로 동적 조정을 통해 coOp과 비슷한 수준의 성능을 달성했다. 즉, 성능이 기대한 바와 다른 이유는 토큰 갯수의 차이에 기인했다고 볼 수 있다.

Discussion

- coOp과 cocoOp의 비교를 통해 토큰 수가 4배임에도 불구하고 cocoOp이 coOp과 비슷한 성능을 냈음을 확인할 수 있다. 이는 단순히 파라미터를 늘리는 것이 아니라, 동적 조정을 통해 unseen 클래스에서 유효하게 동작함을 말해준다. 즉, 일반화에 더 유리할 수 있다. 다만, 시간이 더 오래 걸리는 트레이드 오프가 일어나고, 사용하는 데이터 셋에 따라 coOp이 더 나은 성능을 보일 수도 있기에 적절히 모델을 선택하는 것이 좋을 듯하다.

더블클릭 또는 Enter 키를 눌러 수정